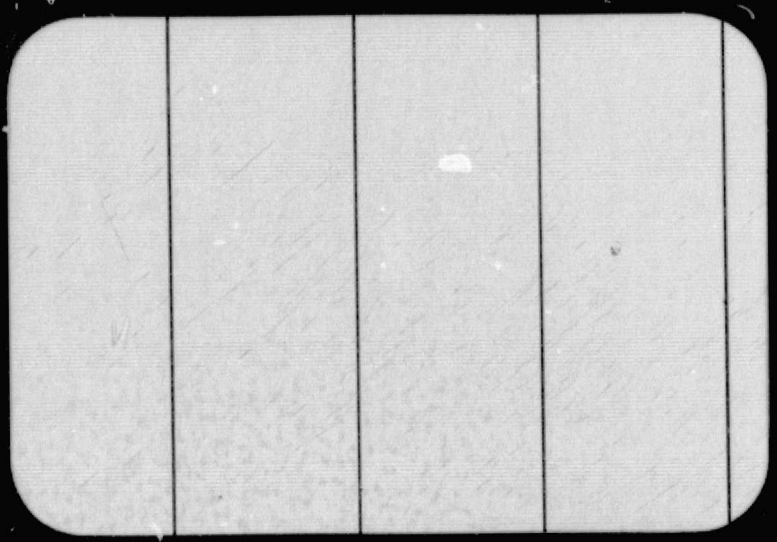
ELECTRICAL ENGINEERING DEPARTMENT



UNIVERSITY OF TENNESSEE

KNOXVILLE TN 37916

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

FINAL REPORT. Contract NSG-5002

Architecture and Data Processing Alternatives for the Tse Computer

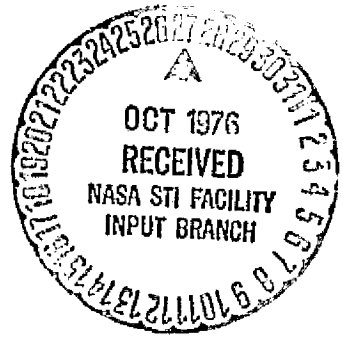
VOLUME 1: Tse Logic Design Concepts and the Development of Image Processing Machine Architectures

D. A. Rickard

R. E. Bodenheimer

TECHNICAL REPORT TR-EE/CS-76-1

September 1976



ARCHITECTURE AND DATA PROCESSING ALTERNATIVES FOR THE TSE COMPUTER

VOLUME 1: TSE LOGIC DESIGN CONCEPTS AND THE DEVELOPMENT OF IMAGE PROCESSING MACHINE ARCHITECTURES

Robert E. Bodenheimer - Principal Investigator
Dale A. Rickard - Co-Investigator
Department of Electrical Engineering
The University of Tennessee
Knoxville, Tennessee 37916

Final Report. NSG-5002
Period: May 1974 - August 1976

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
GODDARD SPACE FLIGHT CENTER
GREENBELT, MARYLAND 20771

ABSTRACT

Schaefer and Strong have proposed a new class of digital computer components which would perform two-dimensional array logic operations (tse logic) on binary data arrays. This dissertation is concerned with the further development of tse logic concepts through the design of Golay transform processing machines that utilize tse logic.

The basic tse components that are currently under development at NASA's Goddard Space Flight Center are described. The properties of Golay transforms which make them useful in image processing are reviewed, and several architectures for Golay transform processors are presented with emphasis on the skeletonizing algorithm.

A hardwired skeletonizing machine is designed using basic tse components. An output disable control line is shown to be an extremely useful addition to active tse logic devices. Two additional hardwired skeletonizing machines are developed using tse logic devices with an output disable control. Several alternate techniques are illustrated for performing the critical index recognition operation, and new tse logic devices are introduced. A unique pipeline architecture is developed for performing ultrahigh speed image processing. In addition, a programmable tse computer capable of performing numerous Golay transforms is designed. Programs are written for performing both the skeletonizing and swelling algorithms.

Conventional logic control units are developed for the Golay transform processors. One is a unique microprogrammable control unit that uses a microprocessor to control the tse computer. The remaining

control units are based on programmable logic arrays.

Performance criteria are established and utilized to compare the various Golay transform machines. On the basis of this research a critique of tse logic is presented, and directions for additional research are identified.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A primary goal of computer architecture research is to increase the speed and efficiency of computing machines. The performance boundary for a particular machine organization is imposed by the basic physical limitations of the available hardware. Therefore, increased computing power must ultimately be obtained through improvements in computer organization. Realizable computer architectures, however, are constrained by the availability of suitable hardware. Thus, optimum computer system development requires that component design and system architecture be considered in concert. This principle is the basis for the general goals of the research reported in this dissertation.

Schaefer and Strong [1]* have proposed a new class of digital computer components which would perform two-dimensional array logic operations (tse** logic) on binary data arrays. The goal of this research effort is to support NASA's development of the tse logic concept through the design of a two-dimensional parallel computer using hypothetical tse logic devices. By developing tse computer system architecture concepts

---

*Numbers in brackets refer to those entries listed in the list of references.

**Tse is the English transliteration of the Chinese word for a pictograph character.

concurrently with the physical hardware, trade offs between tse component complexity and system design constraints can be optimized. For example, useful additions to the tse logic family have been proposed as a result of this research.

Historical Background

The concept of a two dimensional parallel computer was utilized by Unger [2, 3] as a method for improving the performance of digital computers in spatially oriented problems such as pattern detection and recognition. Unger's proposed spatial or SPAC computer consists of a master control unit and a rectangular array of logical modules (Figure 1.1). Each module contains a one-bit accumulator, several one-bit memory registers, and some logical circuitry. Modules communicate directly with their four immediate neighbors. In addition, there is an external input in the form of a photocell. Thus, with the exception of the control unit, a module has all the features of a rudimentary conventional computer.

Unger's SPAC computer utiltizes global control in which identical commands are issued to all logical modules in the array. The master control unit includes a random-access memory for instruction storage, a program counter, and appropriate instruction decoding circuitry. Operation is similar to that of a conventional digital computer control unit except that the control signals are distributed to each module in the rectangular array, rather than to a single arithmetic-logic unit. The global control feature of SPAC limits use of the machine to problems involving applied parallelism.

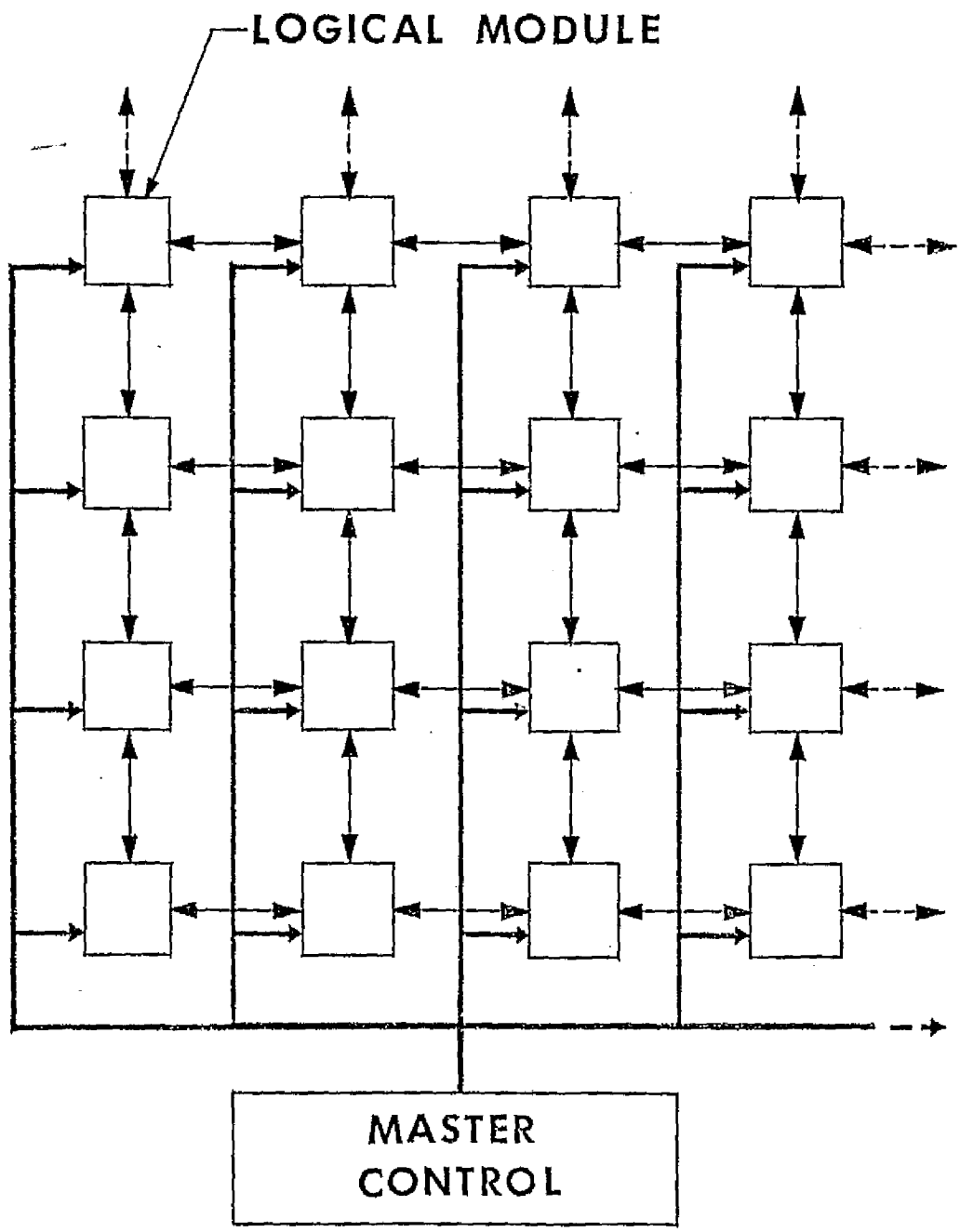


Figure 1.1 Organization of SPAC.

Unger recognized the importance of connectivity in spatial computer operations. As a result, each SPAC module is connected to its eight nearest neighbor modules through special link circuits (Figure 1.2). When a link instruction is executed, a storage register in each link is set if the accumulators of the two logical modules to which the link is connected both contain a one. The storage register is cleared if the accumulator of one, or both, of the modules contains a zero. The state of each link storage register remains fixed until another link instruction is executed.

Expand instructions are used in conjunction with the link operation to perform tasks involving connectivity. For example, a horizontal expand instruction causes a one to be placed in the accumulator of each module which is connected through a horizontal chain of set link elements to two modules with ones in their accumulators. An expansion may be performed with respect to any combination of the four available link orientations: horizontal, vertical, positive diagonal, and negative diagonal. The real significance of the expand operation is that the contents of a module can be directly affected by the contents of arbitrarily distant modules.

Although Unger's computer was impractical in terms of hardware cost, the architecture of SPAC was simulated on an IBM 704 general purpose computer. In this form, the machine was successfully applied to the recognition of alphanumeric characters and the detection of L-shaped patterns.

In 1962 Slotnick [4] proposed a more flexible parallel processing computer called SOLOMON. Like Unger's machine, the SOLOMON computer, shown in Figure 1.3, consists of a large rectangular array of processing

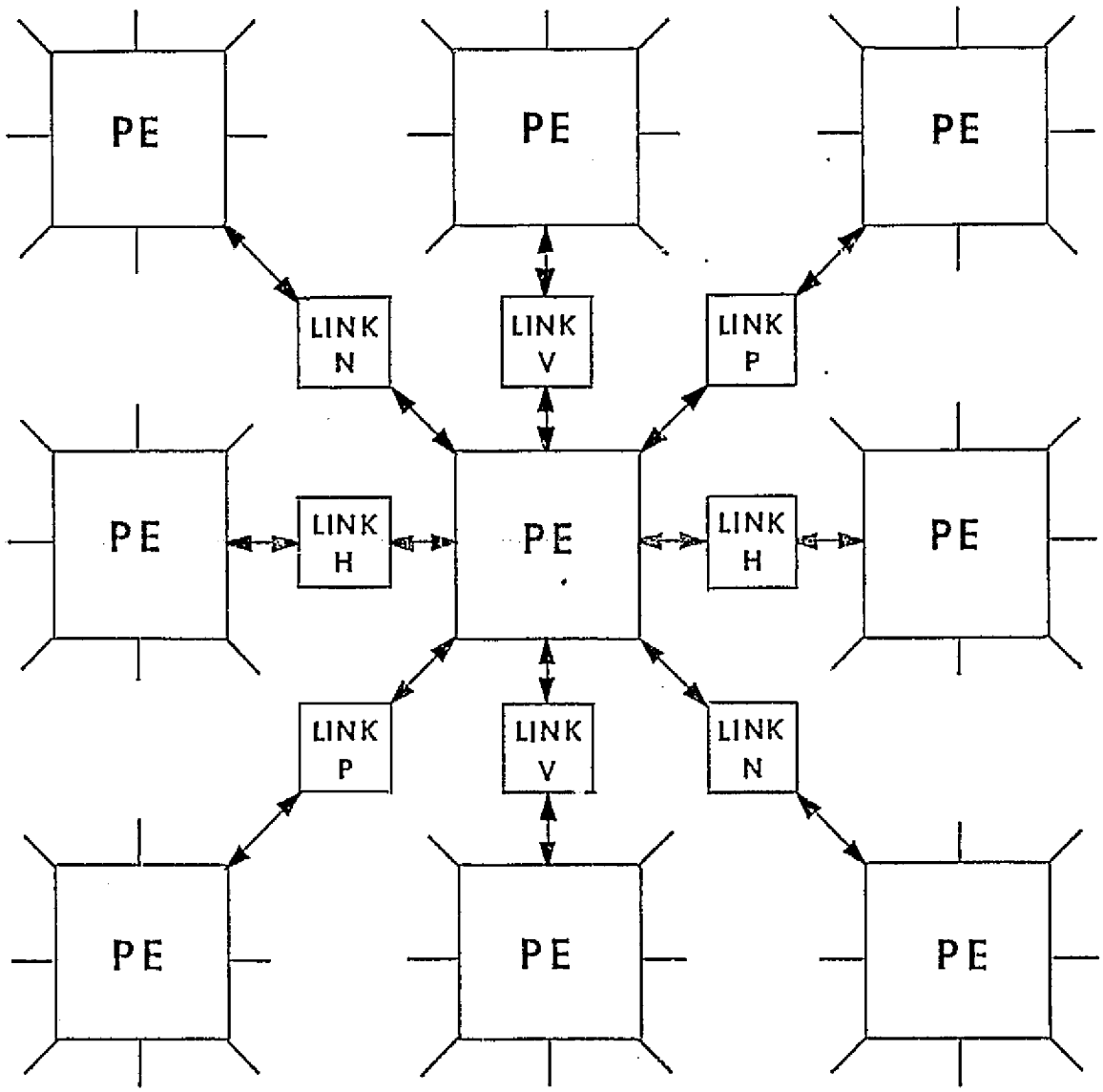


Figure 1.2 Arrangement of link circuits in the rectangular neighborhood of a single processing element.

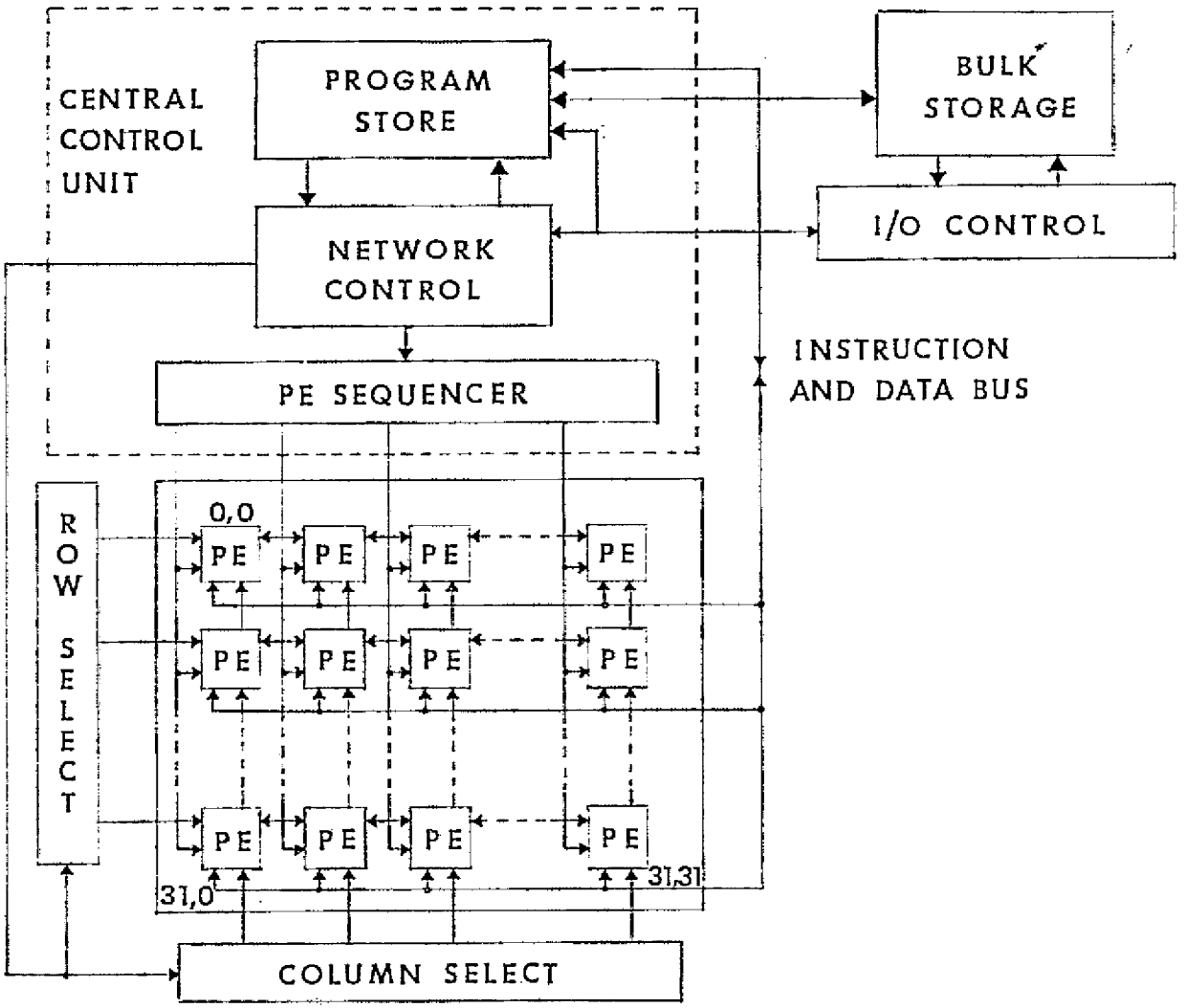


Figure 1.3 Organization of Solomon.

elements (similar to the logical modules in SPAC) and a central, global control unit. There are, however, three basic differences between SOLOMON and SPAC. First, the SOLOMON computer introduces limited local control of each processing element by inhibiting execution of the current instruction wherever the mode of a particular processing element does not match the mode specified by the instruction. A processing element may be in one of four modes as determined by internal conditions and stored data. Thus, although identical instructions are broadcast to each processing element, individual conditional jumps can be programmed [5].

The second basic difference between SOLOMON and SPAC is the intercommunication pattern of the processing element array. Unger's design is based on a simple rectangular array with communication between a module and its four nearest neighbors. The SOLOMON computer retains this basic intercommunication pattern but with five possible modifications. These are as follows:

1. A vertical cylinder formed by establishing communication between the outside columns of the array.
2. A horizontal cylinder formed by establishing communication between the outside rows of the array.
3. A torus formed by combining the first two options.
4. A single straight line formed by a connection of all the processing elements.
5. A circular array formed by connecting the end processing elements of the straight line.

A third distinction between the organizations of SOLOMON and SPAC is the link structure included in SPAC. No equivalent structure was

provided in the SOLOMON computer since SOLOMON was designed primarily for numeric processing.

The SOLOMON design was based on a 32 x 32 array of processing elements. A machine of this size was never constructed due to the excessive hardware cost; however, a 10 x 10 array was built [6]. Research using this machine led to the design of a SOLOMON II computer with a faster clock rate, a faster multiply time, and a 24-bit word length [7]. Further studies and the advent of medium scale integrated circuits encouraged the development of the ILLIAC IV computer which is the largest parallel-array computer now in existence [8].

The arithmetic-logic unit of the ILLIAC IV computer consists of 256 processing elements arranged in four reconfigurable SOLOMON type arrays of 64 processors each (Figure 1.4). A separate global control unit is provided for each array so that the machine can be operated as four independent quadrants, two 128 element arrays, or one 256 element array. Overall system control is provided by a Burroughs B-6500 computer.

Each processing element in the array requires 10^4 emitter-coupled-logic gates to execute 4×10^6 instructions per second. In addition, there are 2048 sixty-four bit words of memory within each processing element. A one giga-bit disk with a transfer rate of 10^9 bits per second is provided for mass data storage. Because of economic considerations, only one quadrant of the four quadrant ILLIAC IV system was originally constructed. Additional details of the ILLIAC IV design and the applications research which has been undertaken using this machine can be found in the engineering literature [5-9].

In 1969 Golay proposed a two-dimensional computer to perform hexagonal parallel pattern transformations using the hexagonal module array

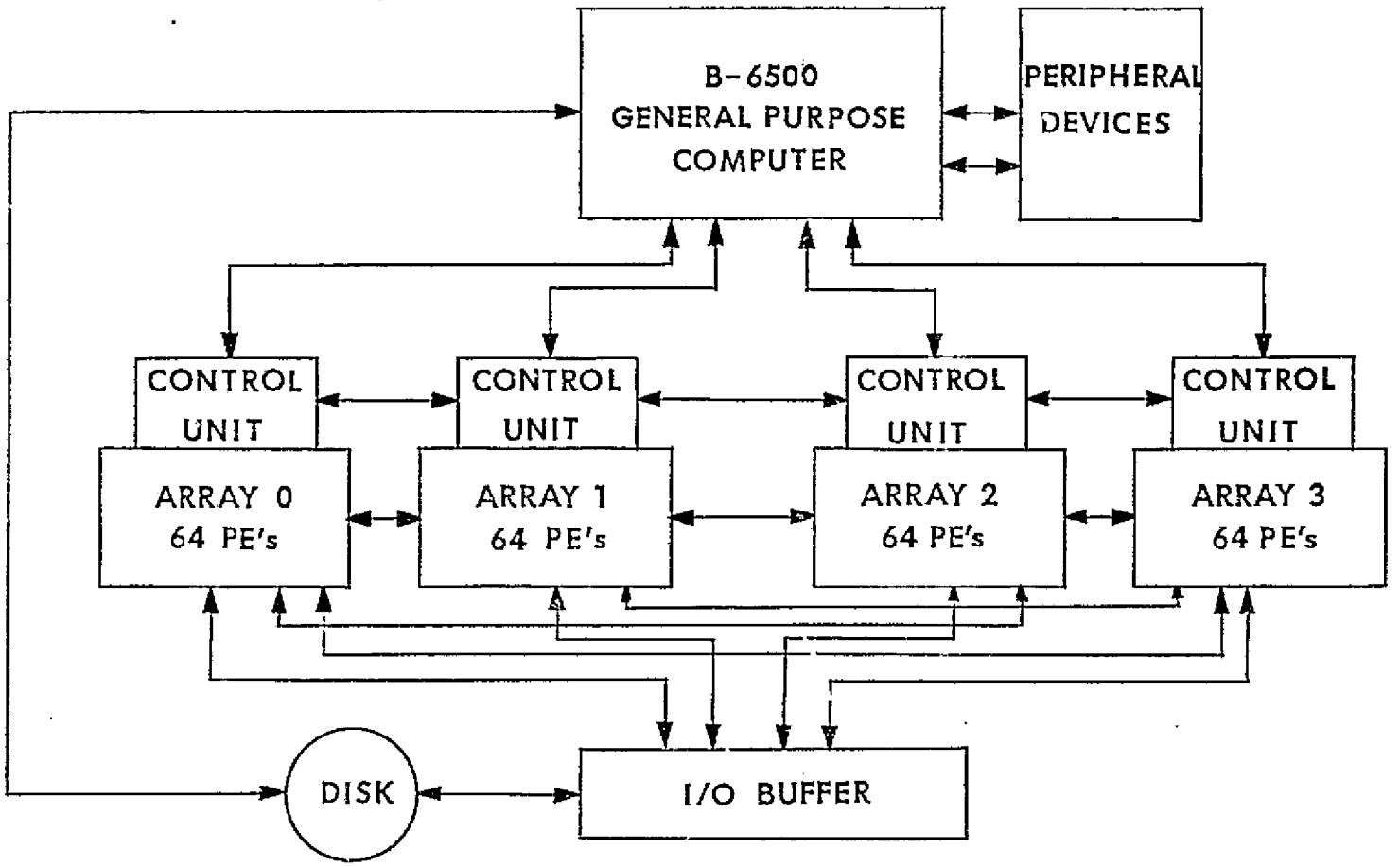


Figure 1.4 Organization of ILLIAC IV.

shown in Figure 1.5 rather than the traditional square module array [10]. The hexagonal tessellation simplifies connectivity dependent operations since each point in the array has six equally distant neighbors rather than four primary neighbors at the distance unity and four secondary neighbors at the distance $\sqrt{2}$ as in the square module array. In addition, the Golay transformations specify division of the array into subfields of non-neighboring modules (Figure 1.6) so that no two neighboring modules need be operated on simultaneously as a function of each other.

The basic computational unit in Golay's proposed computer is the submodule which corresponds to a module in Unger's SPAC computer. Each submodule represents the state of one point in a two-dimensional binary image, and the entire image is represented by a planar, hexagonal layer of submodules. Normally, the machine consists of a number of layers which overlay each other to form a hexagonal array of modules. All of the submodules within one module are interconnected and, furthermore, each submodule of a particular layer, k, communicates directly with its six nearest neighbor submodules within layer k.

The Golay transform classifies the 64 possible patterns of the six element surround of a submodule into the 14 characteristic indices shown in Table 1.1. Submodule operations can be a function of the subfield select signal, the state (1 or 0) of the submodule, the index of the submodule's surround, or the central control unit commands. Often, the modular operations are repeated until no further change occurs in the layer, k, on which the operations are being performed. This condition is detected by forming the modulo 2 sum of the current and the immediately preceding iterations in an auxiliary layer, x, and then determining whether or not layer x is empty.

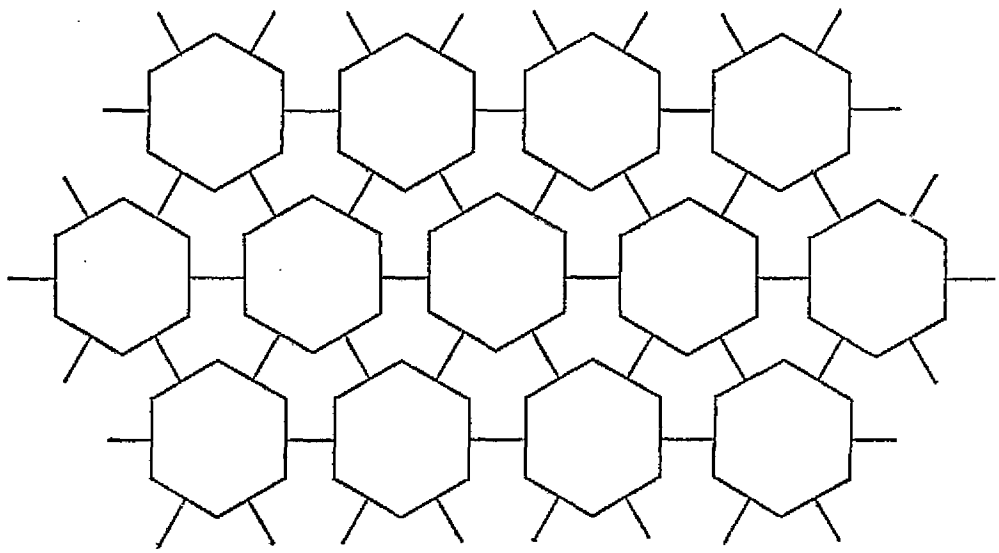


Figure 1.5 Arrangement of Golay's proposed hexagonal module array.

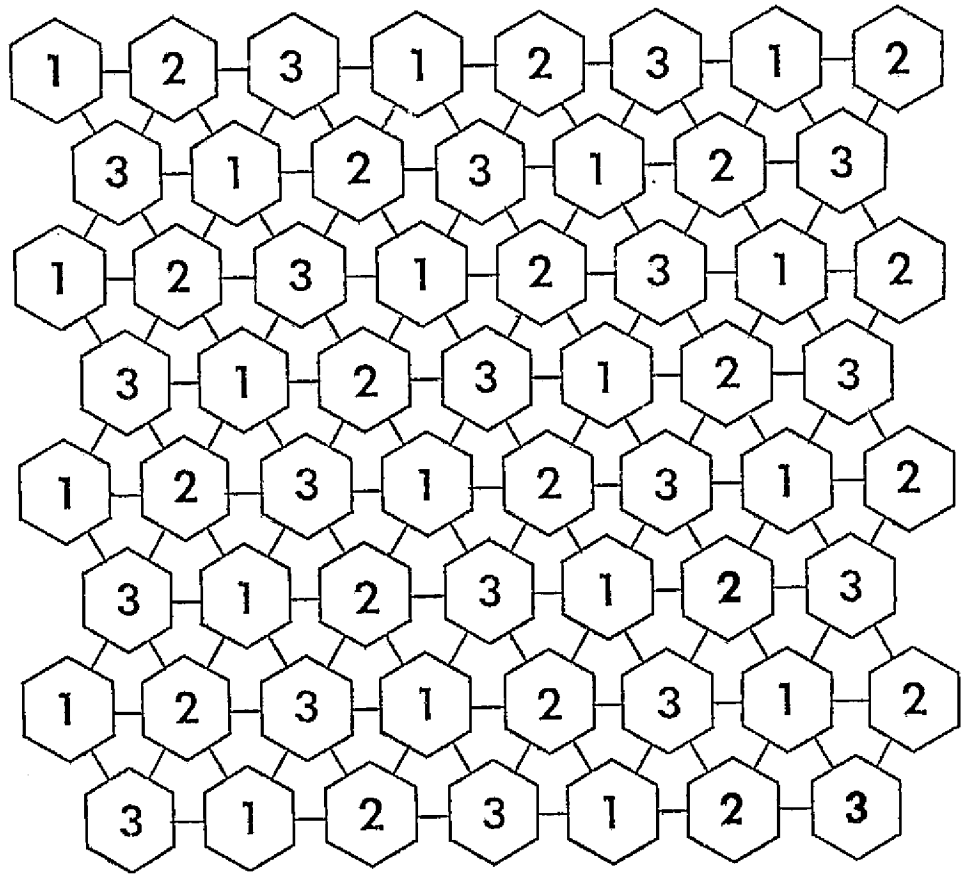


Figure 1.6 Golay neighborhood pattern for a hexagonal array partitioned into three subfields.

TABLE 1.1

GOLAY NEIGHBORHOODS

Pattern	0 0 0 + 0 0 0	0 1 0 + 0 0 0	1 1 0 + 0 0 0	1 1 1 + 0 0 0	1 1 1 + 0 1 0	1 1 1 + 0 1 1	1 1 1 + 1 1 1
Index	0	1	2	3	4	5	6
Weight	1	6	6	6	6	6	1 $\Sigma = 32$
Pattern	0 1 1 + 0 0 1	1 1 0 + 0 1 0	1 1 0 + 0 0 1	1 1 1 + 0 0 1	0 0 0 + 1 1 0	0 1 0 + 0 1 0	1 0 1 + 1 0 1
Index	7	8	9	10	11	12	13
Weight	2	6	6	6	6	3	3 $\Sigma = 32$

Although the two-dimensional computer proposed by Golay was never constructed, a special purpose computer system capable of performing simple Golay transforms on a three layer, 128 x 128 array was implemented [11]. This Golay logic processor (GLOPR) was used successfully to distinguish between two types of white blood cells and to perform other pattern recognition tasks.

Kruse [12] proposed a parallel picture processing machine with many features similar to those of the Golay logic processor but utilizing a rectangular module array. Each module within the array is a synchronous sequential circuit (Figure 1.7) which has a state transition function that is dependent on the present states of that module, the eight nearest neighbors of that module, and a set of global control signals. The number of possible states which a module may possess is limited by design economy since,for even a small number of states, the number of possible neighborhood patterns becomes large.

Each neighborhood pattern is called a template. State transitions occur when the neighborhood of a module matches one of the templates specified in a particular instruction. Kruse allows for rotationally symmetric and iterative operations similar to those used in the Golay transform. "Don't care" states within the neighborhood pattern and limited arithmetical operations are also proposed. The potential problem of conflicting state transitions is avoided by forcing the template matching operation to proceed in a specified order. Only the state transition indicated by the first matching template is allowed to occur.

Two essentially serial, information extracting operations are employed by Kruse's parallel picture processing machine. The first is

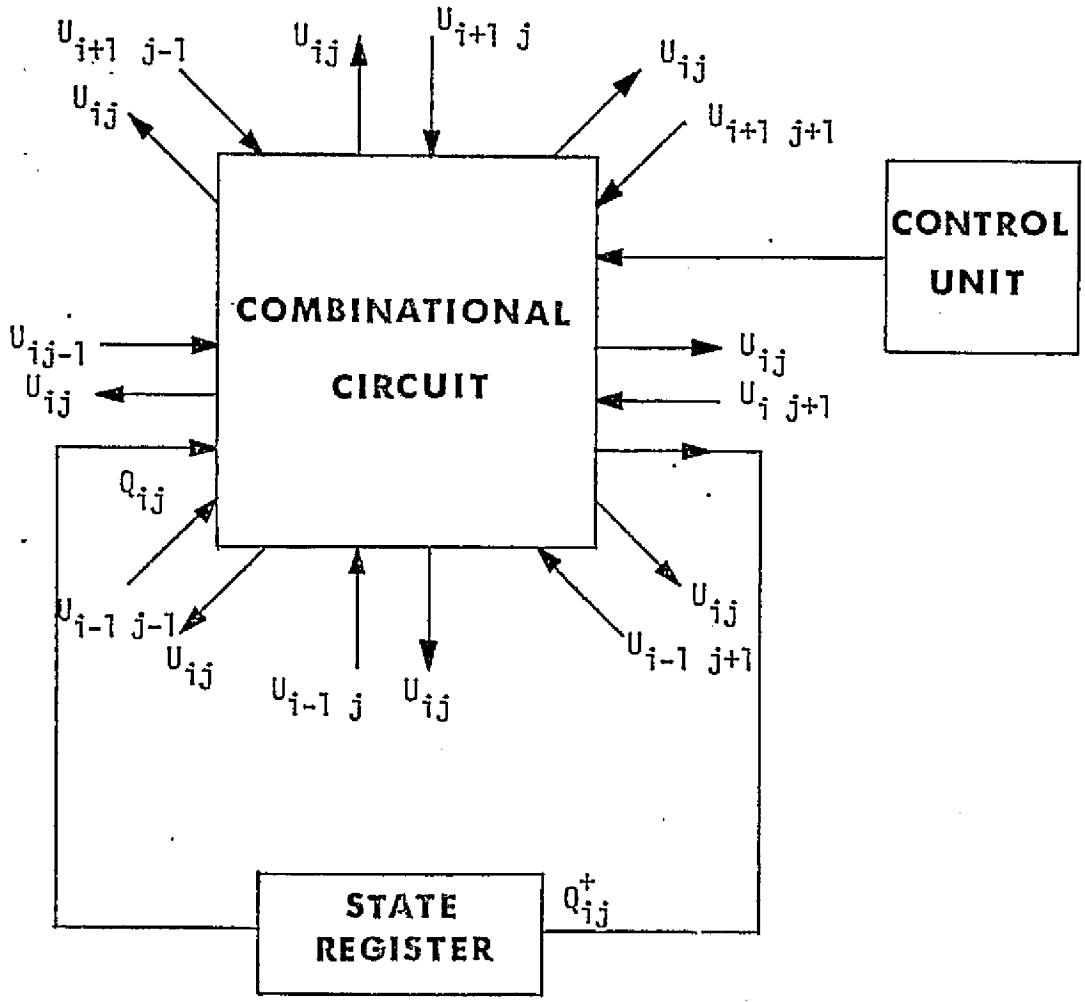


Figure 1.7 A module of the parallel picture processing machine proposed by Kruse.

a neighborhood counting operation in which the number of occurrences of a specific neighborhood is computed during each local operation. This information is used for texture analysis and area measurements. The second is a coordinate extraction operation which identifies the coordinate of the first module encountered in a predetermined scanning sequence whose neighborhood matches a specified template. This process can be used to locate particular features within a picture.

Since all eight nearest neighbors as well as multiple state transitions are utilized in Kruse's machine, the logical modules must be quite complex. Therefore, Kruse did not attempt to present a truly parallel implementation of the design. Instead, a special purpose serial machine capable of performing one local operation at a time was constructed (Figure 1.8). This machine simulates the parallel picture processing operation by sequentially matching templates to each neighborhood in the image. A conventional computer provides system control.

One of the most recent parallel image processing machines, CLIP 3, was described by Stamopoulos [13]. CLIP 3 contains an iterative, 16 x 12 array of logic cells. As shown in Figure 1.9, each cell includes a summation and threshold device, an OR gate, and a function generator. A global control unit provides three control lines that select one of eight threshold levels and eight control lines that select the neighbor inputs which are to be summed. Either a square or hexagonal tessellation can be selected under program control.

The output of the threshold unit is ORed with the contents of the B storage register and applied to one input of the function operator. Storage register A provides the second function generator input. Eight control lines select the Boolean operations which are performed on the

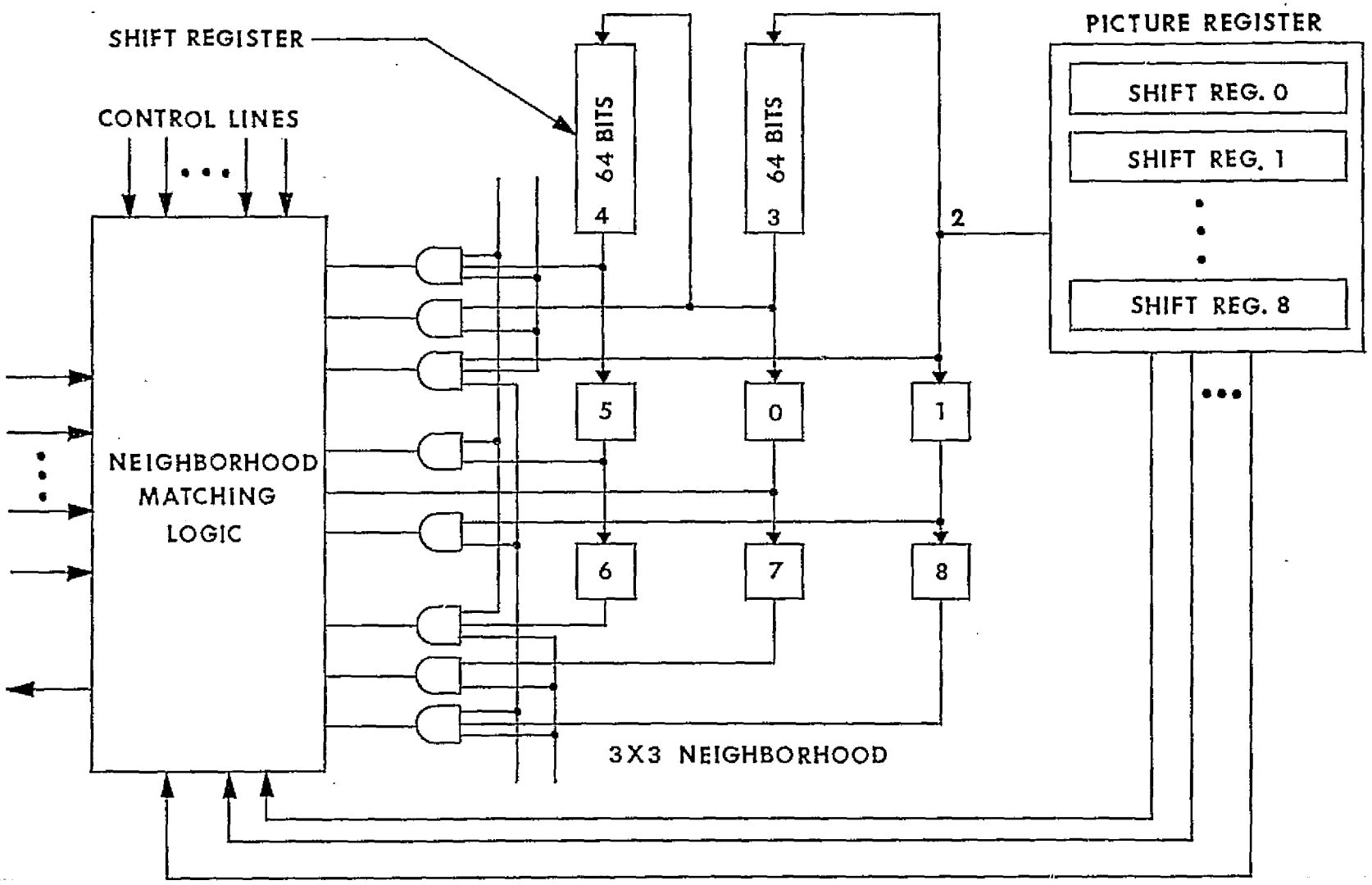


Figure 1.8 Logic unit organization for the serial model of the parallel picture processing machine.

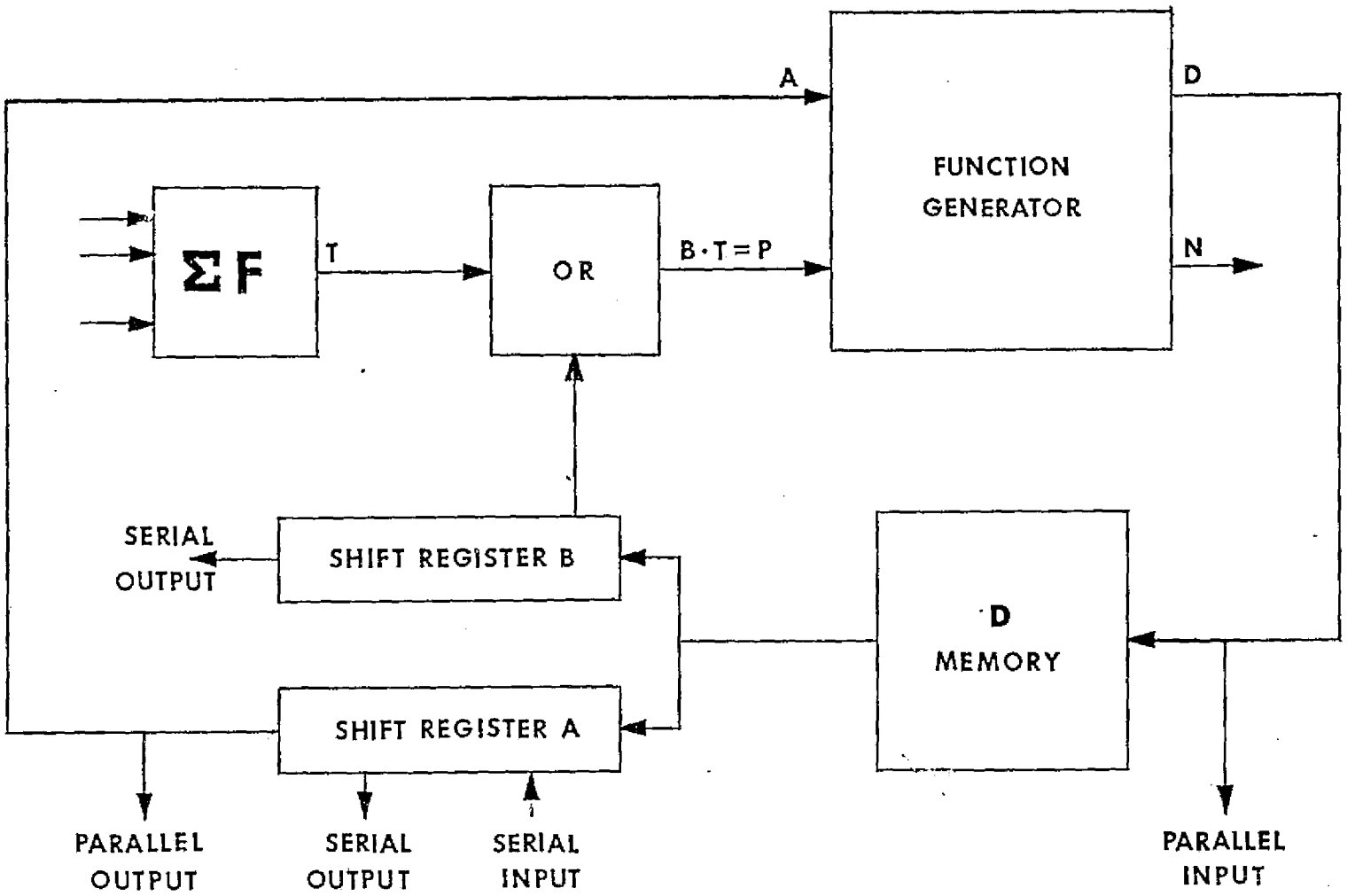


Figure 1.9 Organization of a cell of CLIP 3.

inputs to produce two outputs, N and D. The N output, which represents the state of the cell, is supplied to the threshold units of neighboring cells.

The CLIP 3 processor has been constructed using TTL logic and MOS memory at a cost of approximately $10,000. Serial scanning and display devices are used for economic reasons. The A and B storage devices are shift registers whose contents can be displayed on a dual beam oscilloscope. Pattern inputs are supplied by means of a light pen. Several examples of successful software developed using CLIP 3 are given in [13]. A major limitation of this machine is the relatively small number of cells in the array.

Future Requirements

The two-dimensional parallel computers described in the previous section are potentially orders of magnitude faster than conventional computers in image processing and matrix manipulation tasks. Nevertheless, even more powerful machines will be required to process the 50,000 images per day that NASA expects to receive from earth observation spacecraft during the 1980's [1]. In fact, other important tasks such as real-time modeling of the weather through studies of ocean currents and atmospheric conditions are already awaiting the advent of sufficiently powerful parallel processing machines [14].

In order to provide adequate speed and resolution, future machines may require arrays containing as many as 1024 x 1024 elements. Construction of these machines using hardware organizations such as those described in this chapter would be a formidable task. Although individual

modules with sufficient capabilities could probably be produced as single integrated circuits (e.g., microprocessors), the problem of interconnecting a million or more components to form the complete array would remain. The development of inherently parallel tse logic devices as proposed by Schaefer and Strong [1] is an attempt to make large two-dimensional parallel computers a reality.

A summary of the basic tse logic devices proposed by Schaefer and Strong [1] is presented in Chapter 2. The neighborhood considerations which directed this investigation toward implementation of the Golay transform skeletonizing algorithm are discussed in Chapter 3. Chapters 4 and 5 present basic tse logic implementations of the skeletonizing algorithm. Several new tse logic devices and a cost function are proposed. A hardwired, conventional logic control unit for the tse logic processor is developed. In Chapter 6, a high-speed implementation of the skeletonizing algorithm, based on a unique application of the pipeline principle, is discussed. A programmable tse computer organization which can perform Golay transform algorithms is developed in Chapter 7. A microprocessor based control philosophy is proposed. Chapter 8 summarizes the significant results of this research and suggests directions for future investigations.

CHAPTER 2

BASIC TSE LOGIC DEVICES AND CONCEPTS

Previously, researchers have proposed two-dimensional parallel computer architectures based on planar arrays of logical modules. Each module is essentially a highly specialized microprocessor which, to obtain maximum processing speed, must have a different organization for each unique computer architecture. The tse logic devices proposed by Schaefer and Strong [1] represent an alternate technique for constructing large, two-dimensional parallel computers. This chapter summarizes the work of Schaefer and Strong as reported in [1].

Tse Logic

A tse is a two-dimensional, rectangular matrix of binary data. Tse devices execute logic operations simultaneously on all sets of binary data from corresponding matrix positions within a group of input tses. The primitive operations AND, OR, NEGATE, and SLIDE demonstrated in Figure 2.1 form a functionally complete tse logic set. The AND, OR, and NEGATE operations represent the standard Boolean functions as applied to tses rather than bits, whereas the SLIDE operation translates a binary image an integer number of matrix positions in the ± x and/or ± y direction. Through the four basic SLIDE operations (right, left, up, and down), tse logic provides a unique method of transferring data from any tse matrix position (i,j) to any other matrix position (m,n). Any Boolean function of arbitrarily selected tse data points can be generated

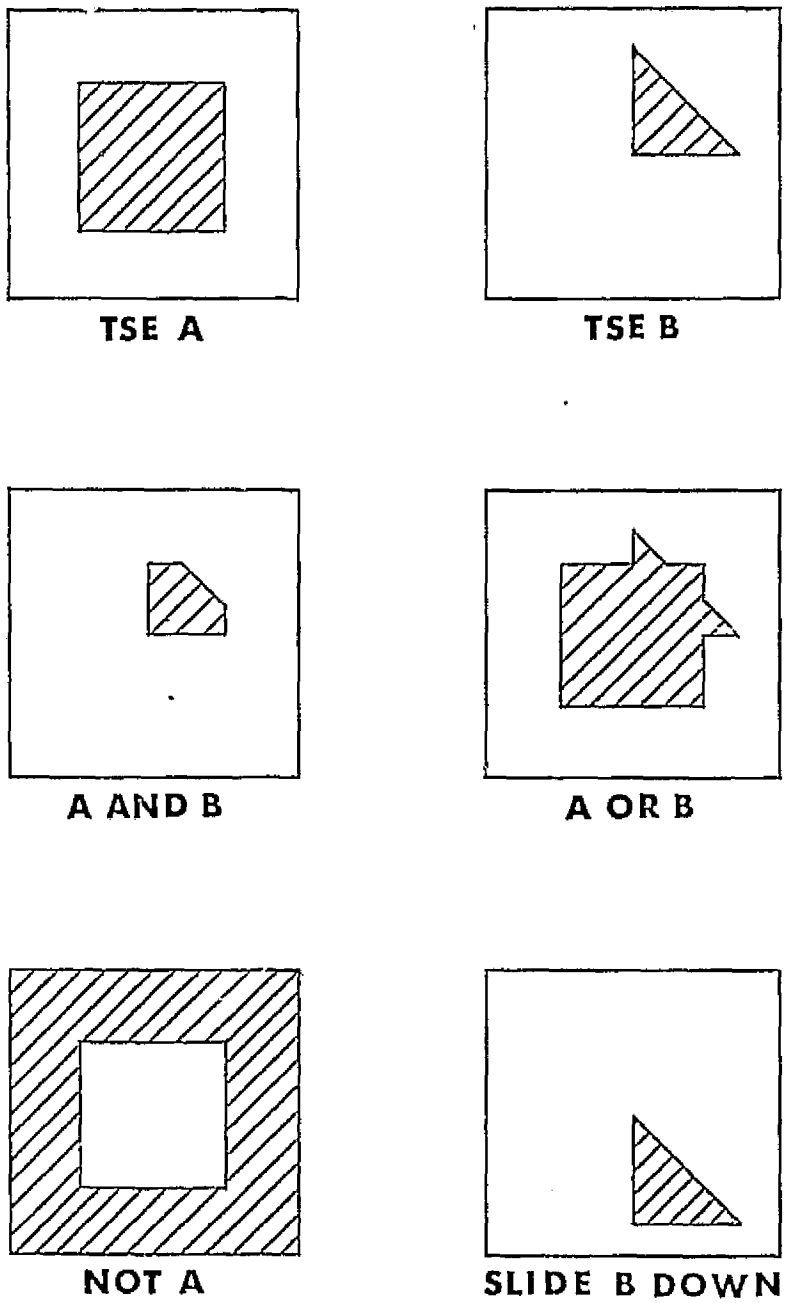


Figure 2.1 An example of primitive tse operations on binary images.

using the primitive tse operations. Therefore, any digital computer can be constructed from a standard set of tse logic devices which perform only elementary operations. The simplicity of the computational cells within each tse logic device is expected to permit their successful integration into the required rectangular array.

Electro-Optical Tse Logic

Schaefer and Strong [1] proposed an implementation of the tse logic concept that is based on a combination of electronic and fiber optic technologies. The AND, OR, and NEGATE functions, as well as a special REFORMAT operation, are performed by active semiconductor devices. SLIDE operations and general data transfers are accomplished using passive fiber optic devices.

Active tse logic devices consist of an integrated array of computational cells. Each cell contains photo-detectors which convert the optical input into electrical signals. The electrical signals are processed by conventional MOSFET circuits and converted to an optical output by either exciting or failing to excite an electroluminescent material. Thus, the state of each data point in a tse is indicated by the presence (state 1) or absence (state 0) of light at the corresponding array position.

Photon coupling between tse logic devices is provided by coherent fiber optic bundles which act as tse image data paths. Optical inputs to individual cells of multiple tse input devices, such as the AND and OR gates, are projected onto the integrated circuit by a fiber optic interleaver. The glass fibers which make up the interleaver are arranged

so that data points from corresponding matrix positions of two input tses are brought into adjacency at the output, thereby becoming inputs to a single cell in the active tse logic device. Initially, the complexity of the interleaver fiber optic array will limit the number of tse inputs to two. The basic tse logic devices are illustrated in Figure 2.2, and schematic symbols for tse devices are listed in Appendix A.

The problem of distributing light to multiple fiber optic bundles restricts the basic fan-out of each tse logic device to one. However, the interleaver can be used in reverse as an image duplicator. Half of the light output of each electroluminescent point in the array appears at each duplicator output. The two modes of operation for an interleaver are illustrated in Figure 2.3, and a prototype interleaver is shown in Figure 2.4. Because of the arrangement of the glass fibers within the interleaver and the decreased light intensity at each output, an active integrated circuit refomatter is normally required at each output. The reformatter serves as a buffer and restores the proper signal levels. Therefore, the effective fan-out of a tse logic device can be increased to two by adding a duplicator (interleaver) and two reformatters to the basic device. Further increases in fan-out can be obtained by cascading additional duplicators and reformatters. As shown by Schaefer and Strong [1], the reformatters can be replaced by negators when the complement of the input tse is required. This approach reduces propagation delay and the number of components required in some tse circuits but also reduces the potential noise immunity of the negator device since the logic one input threshold must be set to less than one-half of the normal logic one light intensity. The basic tse logic circuit for the EXCLUSIVE-OR of two tses is illustrated in Figure 2.5.

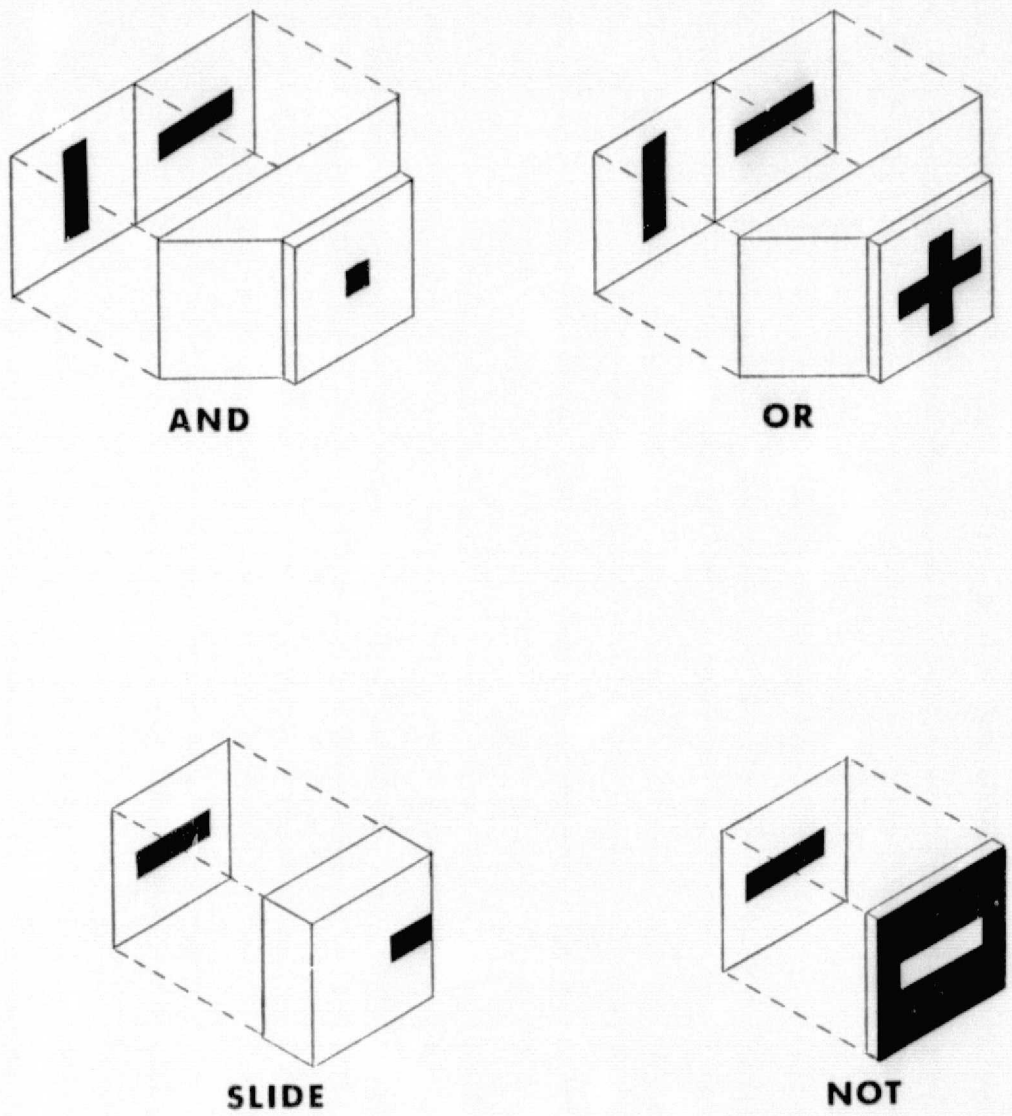


Figure 2.2 Basic tse logic devices.

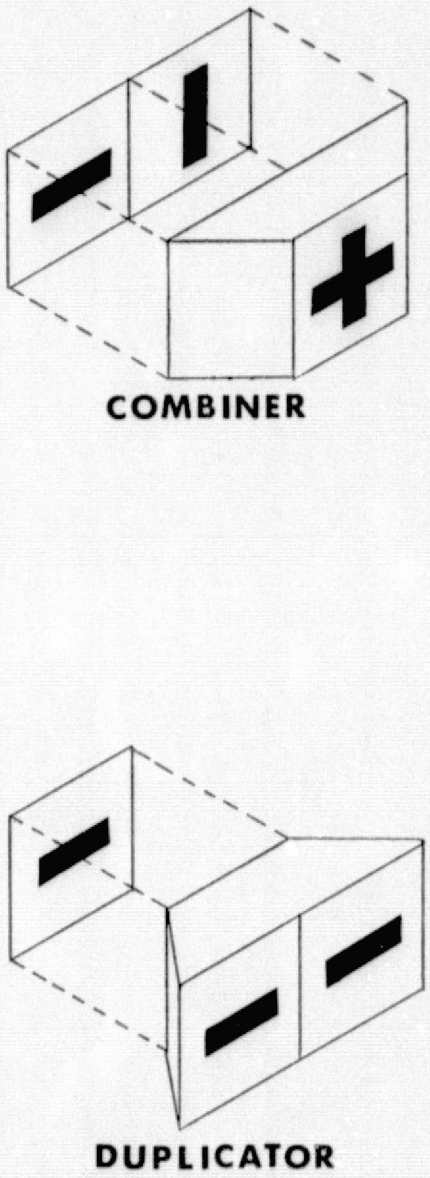


Figure 2.3 Two modes of interleaver operation.

Figure 2.4 A prototype interleaver.

(Courtesy of Earth Observation Systems Division, Goddard Space Flight Center)

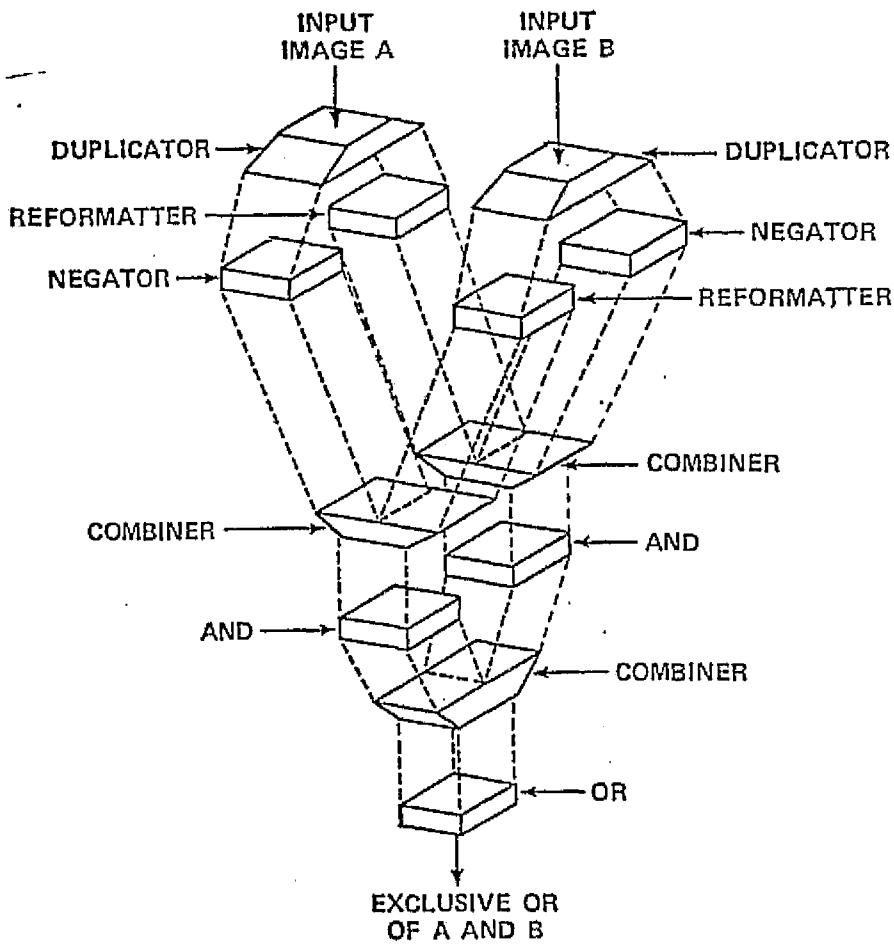


Figure 2.5 Exploded view of a tse EXCLUSIVE-OR circuit. (Courtesy of Earth Observation Systems Division, Goddard Space Flight Center).

SLIDE operations are accomplished by transferring the tse from one fiber optic image path to another path which is offset in the x and/or y direction. Normally fibers in the output path which receive no inputs from the original tse are masked in order to guarantee a logical zero in the corresponding position of the output tse. Figure 2.6 demonstrates the image path configuration required for a SLIDE down operation. By utilizing the proper offset, a SLIDE gate can be constructed so as to translate a tse any number of matrix positions.

SLIDE gates are one of several intercommunication devices proposed by Schaefer and Strong [1]. The other intercommunication devices being considered for use in tse computers include: a cycler, a rotater, a vertical inverter, a horizontal inverter, a diagonal inverter, a horizontal sweeper, a vertical sweeper, a contractor, a spiller, a magnifier, and a demagnifier. In the context of this study, the most important special intercommunication devices are the contractor and the spiller. A contractor has one tse input and one optical output element. The output signal is a logic one if any elements of the input tse are in the logic one state. This device is useful for detecting an empty or all zero tse which corresponds to a black image. Two potential realizations of the contractor are illustrated in Figure 2.7. The spiller device shown in Figure 2.8 has one tse input and one tse output. The output tse is a white or all logic one image if and only if the element of the first row and first column of the input tse is a logic one. Spiller operation can be obtained from the tse circuit shown in the lower section of Figure 2.7 if all elements of the input image except element (1,1) are masked.

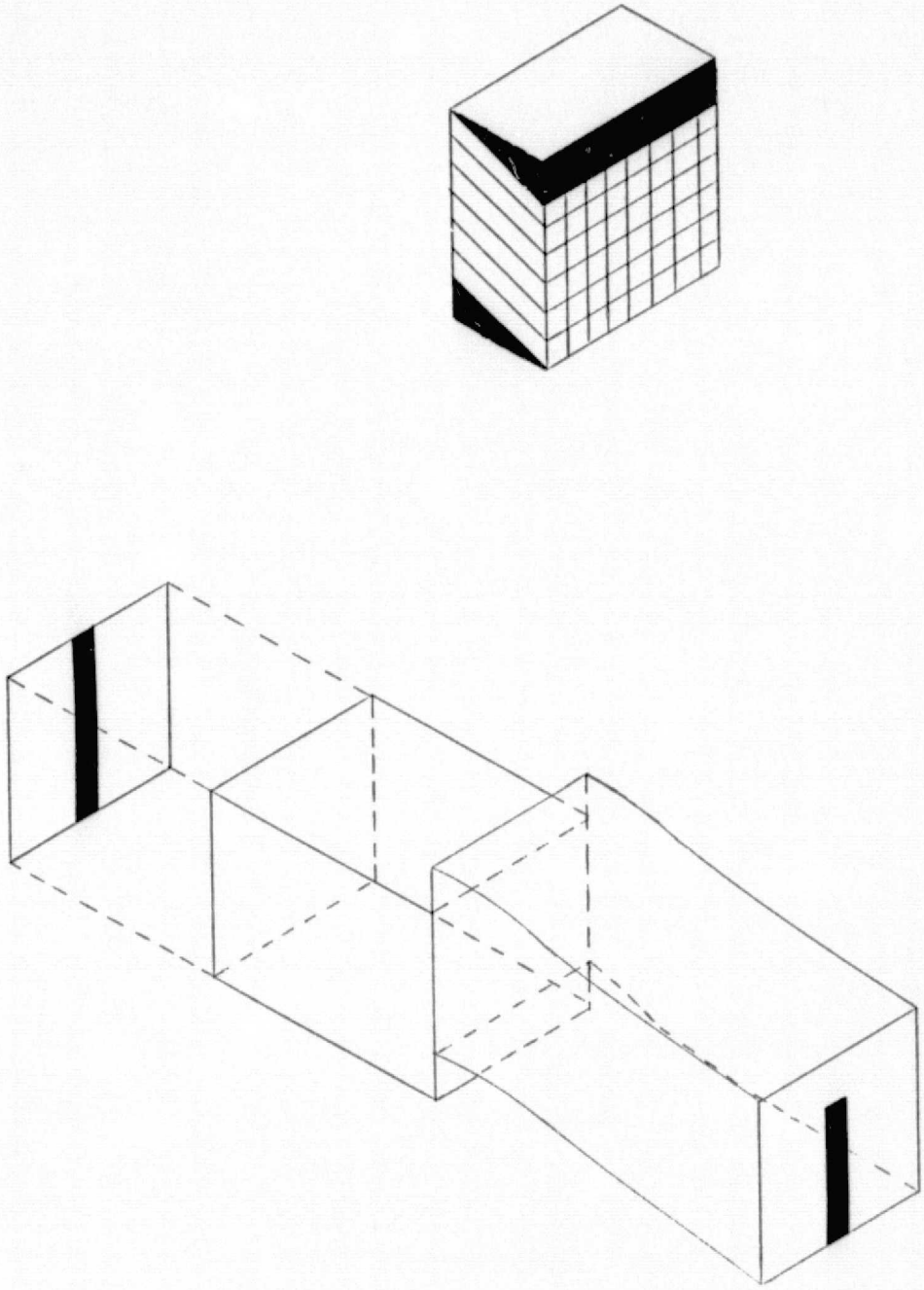


Figure 2.6 Alternate SLIDE down device structures.

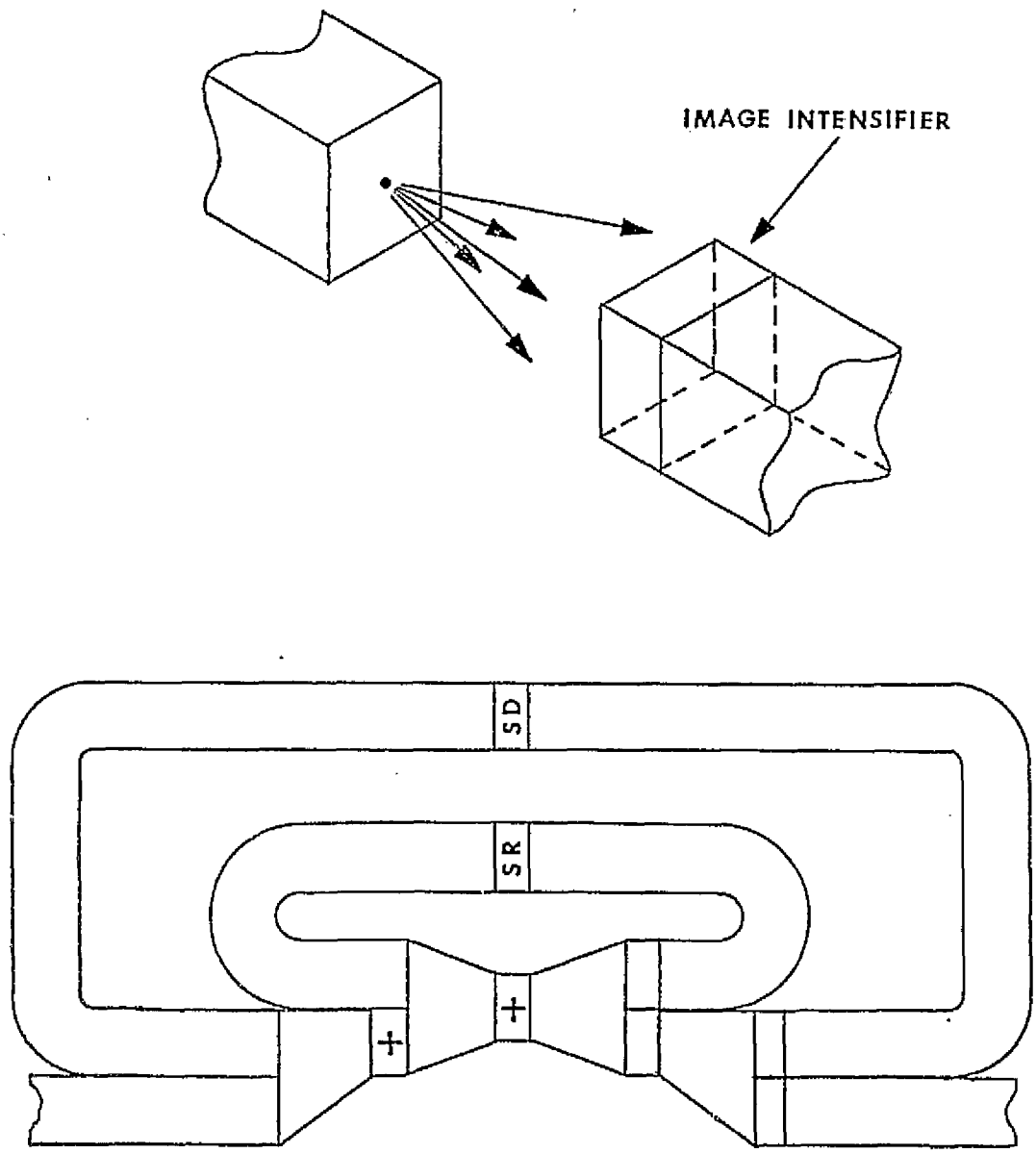


Figure 2.7 Alternate implementations of the contractor device.

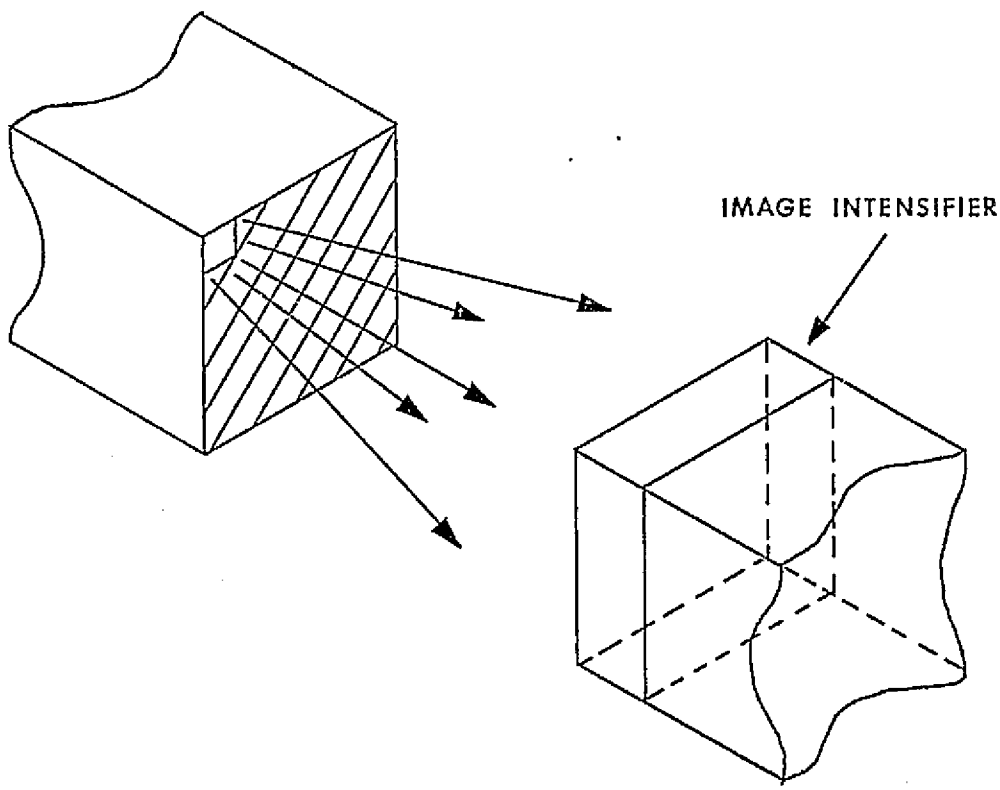


Figure 2.8 An image intensifier implementation of the spiller device.

Performance specifications for the electro-optical tse logic components will be an important consideration in the development of tse computer architectures. The experimental devices currently being developed are based on a 128 x 128 element array. Initially, active tse logic devices will have a response time (propagation delay) of approximately five milliseconds and will consume up to three watts of power. Passive devices will add significant volume and weight to the tse computer structure. For example, the prototype interleaver (Figure 2.4, page 27) weighs 0.7 grams, is six centimeters long, and has an output image area of one square inch. Major objectives in the development of tse logic devices will be to increase the array size to 1024 x 1024 elements while reducing the component size, power consumption, and propagation delay. A major advantage of the tse logic concept is that these improvements can be incorporated into individual active tse logic devices as the electro-optical technology improves.

Tse Analog-to-Digital Conversion

A major application of tse logic will be processing the multiple gray level images received by earth observation spacecraft. Before these images can be processed, they must be digitized into a set of binary tses. Figure 2.9 demonstrates the concept of digitizing an image into three tses by using threshold devices. The output tse data word consists of three ordered tses and represents an eight level quantization of the original image. A six tse data word derived from a 1024 x 1024 pixel image would contain over six million bits of data representing 64 gray levels. Figure 2.10 illustrates the tse logic concept as applied to processing earth resources data.

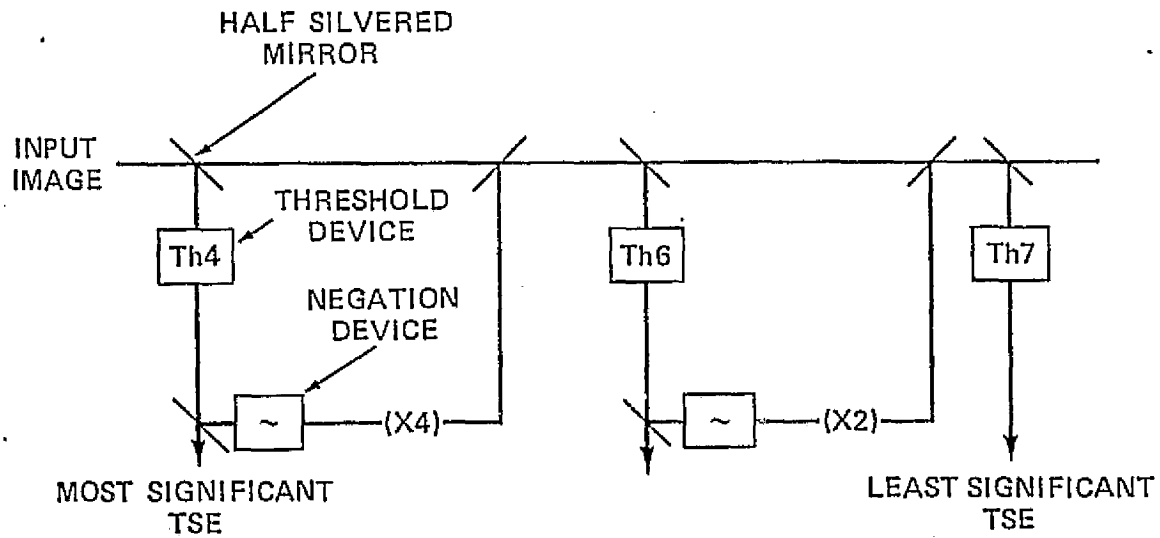


Figure 2.9 Tse analog-to-digital conversion hardware schematic for an eight level image.
(Courtesy of Earth Observation Systems Division, Goddard Space Flight Center)

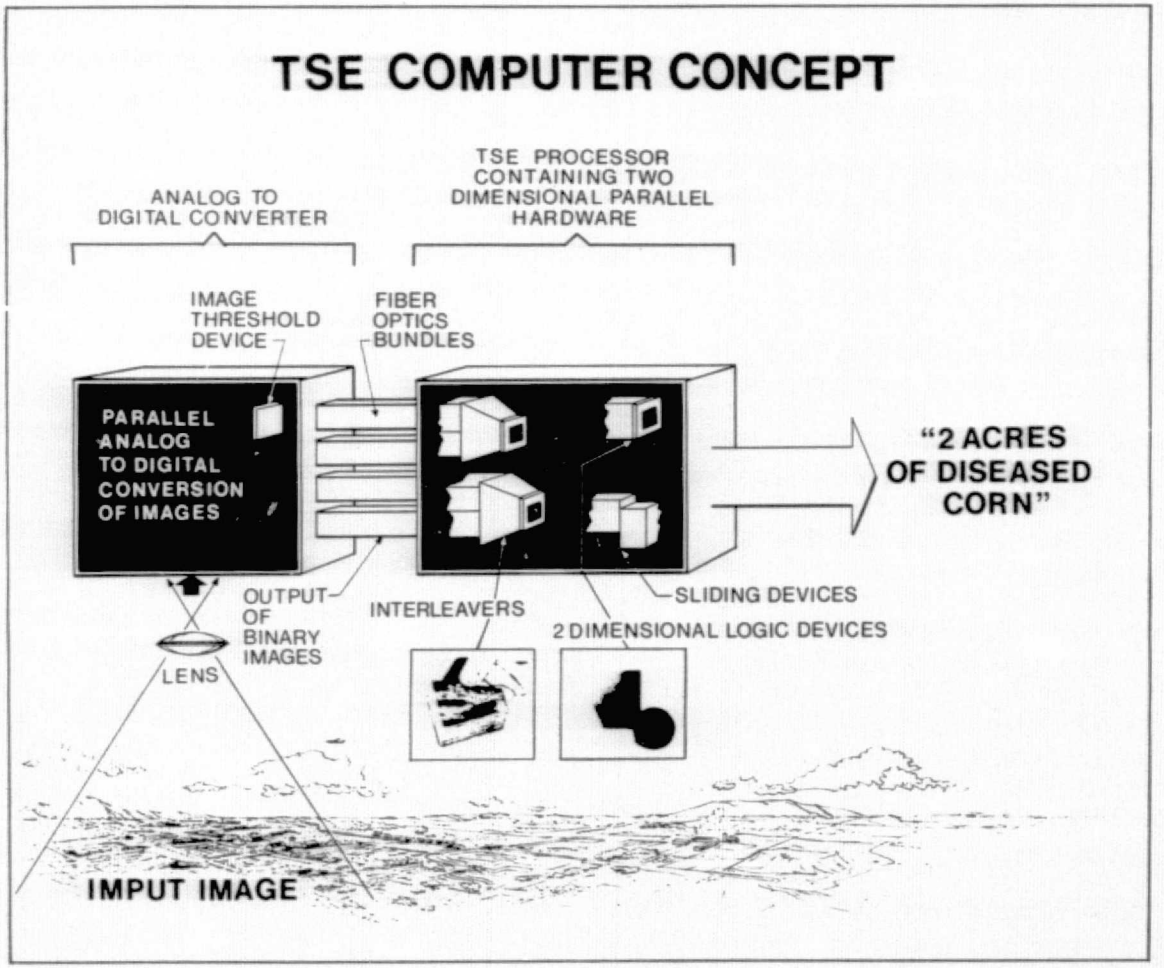


Figure 2.10 Tse computer concept.

(Courtesy of Earth Observation Systems Division, Goddard Space Flight Center)

CHAPTER 3

GOLAY HEXAGONAL PARALLEL PATTERN TRANSFORMATIONS

Many important pattern detection and recognition algorithms are based on nearest neighbor logic in which each point in a binary image is treated as the basis point of a localized neighborhood. These algorithms utilize image transformations which require various operations to be performed on the basis point as a function of the states of the basis point and the points within its neighborhood. The extent and form of the neighborhood must be carefully matched to the application so that hardware complexity can be minimized. In the first section of this chapter, some factors which influence the choice of a neighborhood are discussed. The second section reviews a particular type of pattern recognition algorithm which utilizes nearest neighbor logic and is the basis of the tse logic designs presented in the remainder of this dissertation.

Neighborhood Considerations

As pointed out by Golay [10], there are only three types of planar, symmetric, isotropic point arrangements: the square, hexagonal, and triangular arrays. In a standard, uniformly-shaped rectangular digitization pattern, each basis point has four primary neighbors at the distance unity, four secondary neighbors at the distance $\sqrt{2}$, and additional neighbors at distances of two and greater. The number of points which can be considered in the basic neighborhood is limited by the practical considerations of hardware complexity and cost. Therefore, the simple

eight point neighborhood shown in Figure 3.1 is normally selected for the rectangular array. As shown by Kruse [12], more complex neighborhood operations can be simulated by selected sequences of local operations using the basic neighborhood.

The rectangular neighborhood is advantageous in pattern recognition algorithms which depend on an orthogonal coordinate system. In general, however, the rectangular neighborhood requires more complex processing than the hexagonal neighborhood [10]. As shown by Golay [10], any operation on a basis point, P, which requires knowledge of the connectivity properties of the rectangular neighborhood of P should be a function of both the four primary and the four secondary neighbors of P. However, since the secondary neighbors are at a somewhat greater distance from P, the function should depend less strongly on these variables. Thus, the development and implementation of algorithms involving the connectivity of rectangular neighborhoods is a complex task.

This task becomes even more complex when the triangular array is employed. In order to determine whether or not a basis point operation will alter the connectivity of a triangular array, one must know the states of three nearest neighbors at distance unity, six neighbors at distance $\sqrt{3}$, and three neighbors at distance two. Therefore, the triangular array is not normally used in pattern recognition algorithms.

In contrast, the uniform hexagonal digitization pattern proposed by Golay [10] yields a basic neighborhood (Figure 3.2) of six points which are equidistant from the basis point. As a result, the connectivity properties of the hexagonal neighborhood are readily defined as a function of the basis point and its six nearest neighbors. The single disadvantage of the hexagonal array is that the natural coordinate axes do

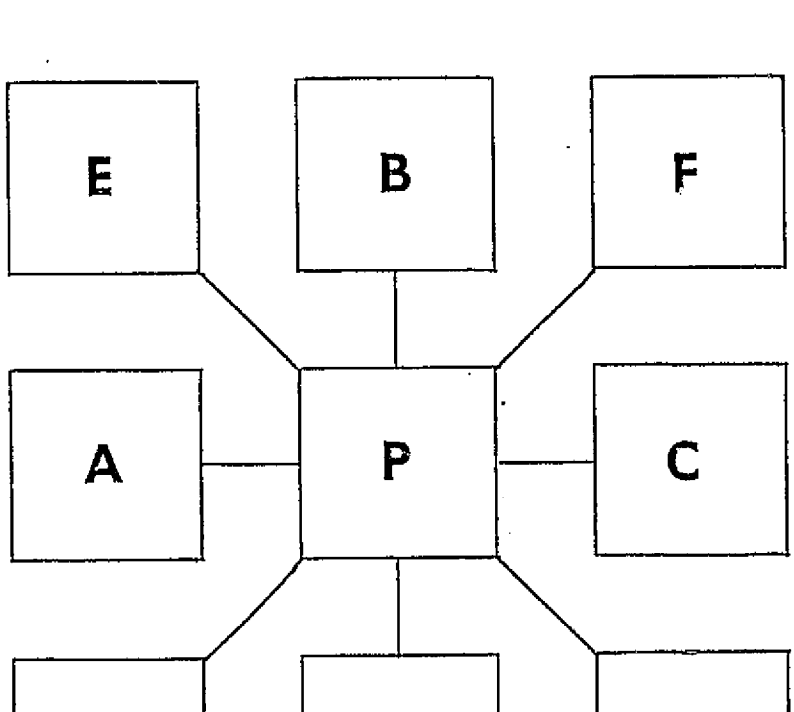


Figure 3.1 Eight point rectangular neighborhood.

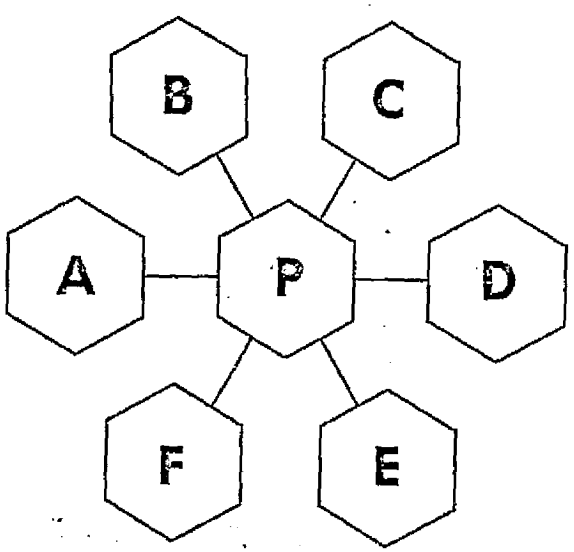


Figure 3.2 Six point hexagonal neighborhood.

not correspond to an orthogonal system. Therefore, the hexagonal array is favored in applications, such as the extraction of earth resources data from satellite pictures, where the required algorithms do not rely on orthogonal coordinates.

Golay Transforms

The Golay transform is based on the set of 14 rotationally independent patterns of zero and one states which can occur in the surround (neighborhood) of a basis point within a hexagonal array. As shown in Table 1.1, page 13, each pattern is assigned a characteristic index and a weight which indicates the number of distinct orientations of the pattern that can be obtained by rotating the illustrated surround. The sum of the weights is 64, the total number of possible neighborhood patterns. Because all patterns of the same index are considered equivalent, Golay transform operations are invariant under rotation of an image through angles which are multiples of 60^{0}.

The connectivity property of a Golay hexagonal neighborhood is uniquely defined by the state of the basis point and the index of the surround. For example, changing the state of a basis point within a neighborhood of index six or less cannot alter the connectivity of the array. However, the connectivity of the array can be altered by changing the state of a basis point with a neighborhood of index seven or greater. A knowledge of both the state of a basis point and the index of its surround is sufficient to determine the effect which a particular pattern transform operation will have on the connectivity of a neighborhood with basis point P, if and only if the neighbors of P are not simultaneously transformed as a function of P. The Golay transform satisfies this

condition by dividing the hexagonal binary data array into a number of subfields, each of which contains only non-adjacent points. Three subfields are commonly used although, as shown in Figure 3.3, the array can also be divided into four or seven subfields. All data points within a particular subfield are processed in parallel. However, only one subfield is transformed at a time to avoid the potential logical conflicts.

Thus far this discussion has assumed a simple Golay transform in which the surround of each basis point of an image, k, is equivalent to the hexagonal neighborhood of that point. The Golay transform also permits more general compound operations in which the surround of a basis point of image k can be defined as a logical function of the hexagonal neighbors of the corresponding points in a multiplicity of other binary images, p,q,...,u,v. The compound Golay transform can be utilized in processing multiple grey level images which have been digitized.

The general hexagonal parallel pattern transformations [10] are basis point operations which are performed simultaneously on all points within one subfield of an image and then sequentially on each subfield. Golay transforms are expressed in the form

$$M_{a_0 a_1 \ldots a_{13}, n}\left[k = a_{i(L_1)} \cdot L_2 \cdot L_3 + \overline{a_{i(L_1)} \cdot L_2} k \right] \qquad (1)$$

where M stands for the general basis point operation. The a_0 - a_{13} subscripts of M represent the indices of the basis point's surround for which an operation will be performed. Subscript n stands for the number of iterations required. Each iteration involves performing the indicated operation on each of the specified subfields in turn. When n is not specified, the operation is performed only once. If n is replaced

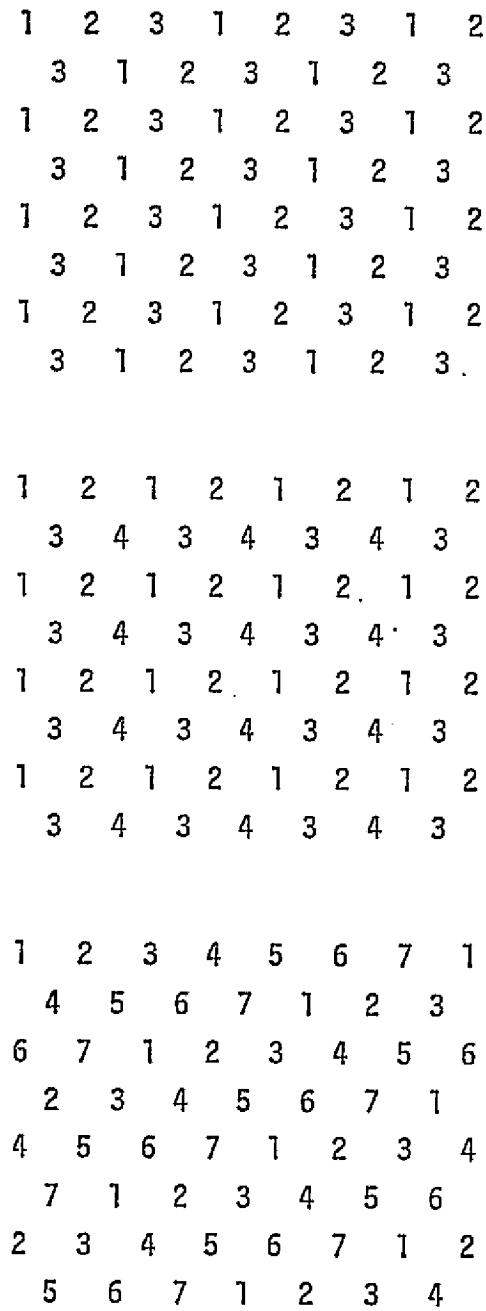


Figure 3.3 Hexagonal arrays of three, four, and seven subfields.

by the symbol ∞ instead of a number, the operation is performed until the image ceases to be transformed by further iterations.

In Equation (1), k represents the binary state of the basis point being operated on. When a simple Golay transform is being performed, L_1 is a logical function of one or more of the hexagonal neighbors of the basis point in image k. In the case of a compound Golay transform, L_1 can also be a function of the hexagonal neighbors of corresponding basis points in images p,q,...,u,v. In general, $L_1 = L_1(k,p,q,\ldots,u,v,\ldots)$ and $L_1 = k$ implies a simple Golay transform. The surround of the basis point consists of the six outputs of the function L_1 which correspond to hexagonal neighbors a-f in the simple Golay transforms. The term $i(L_1)$ is the index of the surround of the basis point on which the operation is being performed. When the index of the surround is listed as a subscript of M, $a_{i(L_1)} = 1$; otherwise, $a_{i(L_1)} = 0$.

L_2 specifies which one of the three, four, or seven subfields is currently being transformed. For an image divided into four subfields, L_2 will be true for only one-fourth of the image points at any one time. When the operation is to be performed on all subfields of an image, a superscript of three, four, or seven may be given with M instead of specifying L_2. L_3 can be either a control signal or a logical function of the various images utilized in compound Golay transforms.

A number of useful Golay transforms are given in [10] and [11]. One example is the simple skeletonizing operation defined by

$$M^{3}_{1\text{-}2,\infty}\left|k = \overline{a_{i(k)}} \cdot k\right|. \tag{2}$$

When this algorithm is applied to a binary image plane, all simple blobs

are reduced to a single logic one point, and all blobs with holes are reduced to one or more loops consisting of a single layer of logic one points. Figure 3.4 demonstrates the application of this algorithm. The superscript of M specifies that the points in the binary image are assigned to one of three subfields by the numbering scheme shown in Figure 3.3, page 42, and that the subfields are processed sequentially in ascending order. The first subscript of M indicates that the basis point operation is to be performed whenever the index of the surround is one, two, or three. Therefore, whenever $i(k) = 1, 2,$ or 3

$$a_{i(k)} = 1 \quad \text{and} \quad \overline{a_{i(k)}} = 0 \tag{3}$$

otherwise

$$a_{i(k)} = 0 \quad \text{and} \quad \overline{a_{i(k)}} = 1 \, . \tag{4}$$

As specified by the second subscript of M, the skeletonizing operation is performed iteratively until the image becomes stable. When each subfield is processed, points within that subfield are set to zero unless they are currently in state one and have a surround whose index is not one, two or three. Figure 3.4 shows only one iteration of the algorithm. However, this simple image will not be transformed further by another iteration, so the final skeleton is the result of the third subfield operation.

Although the skeletonizing algorithm is a simple Golay transform, its tse logic implementation must embody the general Golay transform principles. In addition, hardware minimization is a particularly important consideration in the initial development of tse logic circuits. Therefore, implementation of the skeletonizing algorithm will be

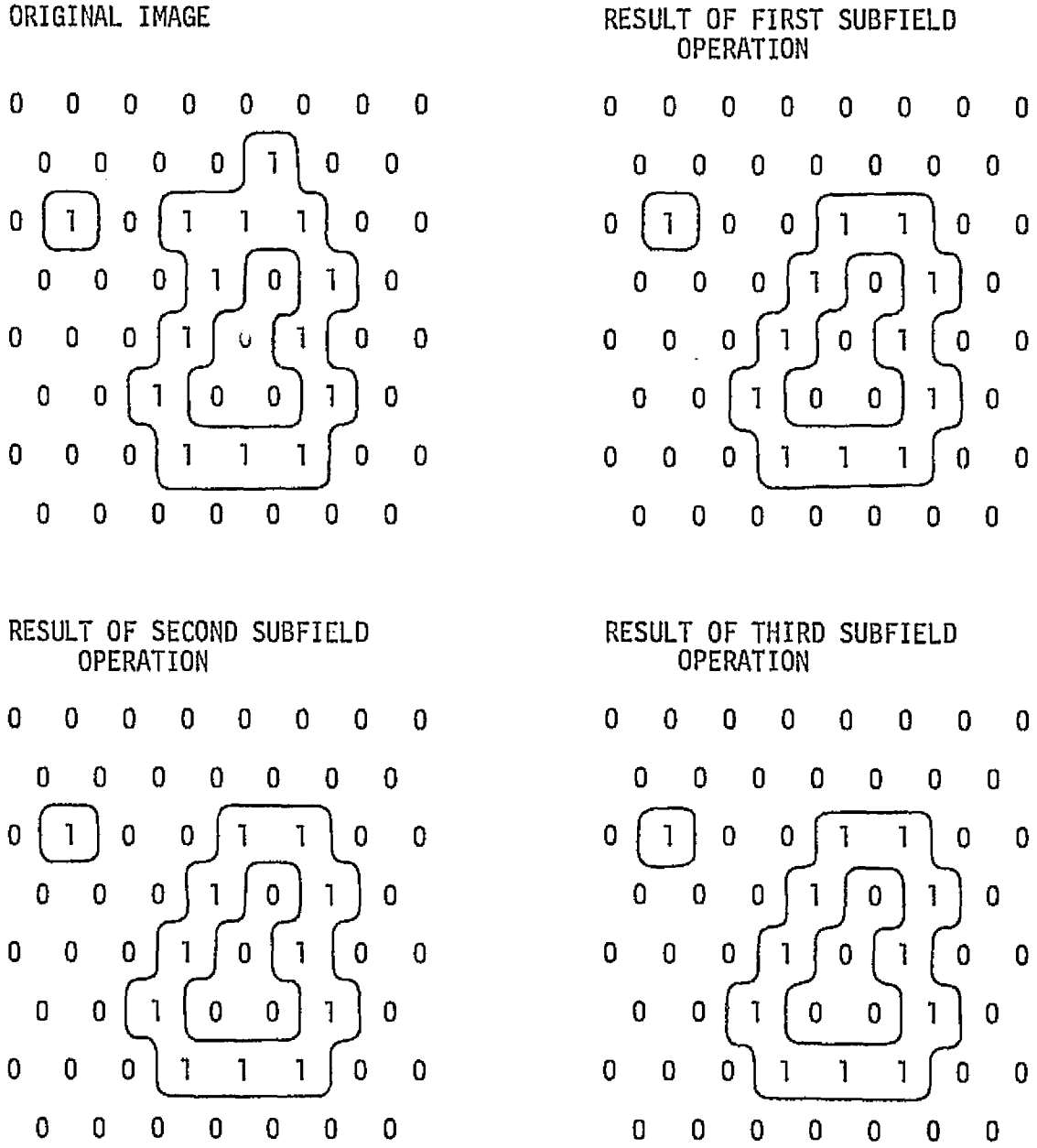


Figure 3.4. An example of the Golay transform skeletonizing algorithm.

emphasized in the tse logic designs presented in the following chapters. Little generality will be lost since, as demonstrated in Chapter 4, any simple Golay transform can be implemented by a straightforward modification of the basic tse logic skeletonizing machine.

CHAPTER 4

BASIC TSE LOGIC DESIGN CONCEPTS AND THEIR APPLICATION TO DESIGN OF A GOLAY TRANSFORM PROCESSOR

Every switching function can be expressed in a canonical sum-of-products form, where each expression consists of a finite number of switching variables, constants, and the operations AND, OR, and NOT [15]. The family of electro-optical tse logic devices proposed by Schaefer and Strong [1] is functionally complete since the AND, OR, and NOT operations are all available. Thus, tse circuits analogous to each of the major subdivisions of a conventional digital computer can be designed. A computer organization in which arithmetic-logic unit, control unit, and memory are all constructed from tse logic devices is conceivable. If such a computer is developed by interconnecting tse circuits which correspond directly to the logic circuits of a conventional computer, the tse computer will simply be a two-dimensional expansion of the conventional computer. Each element in the n x n array of a tse logic device becomes one component of an elemental computer which is isomorphic to the original conventional computer [1]. This organization, however, does not realize the full power of tse logic since tse intercommunication devices provide the potential for a more sophisticated design in which data can be transferred between the elemental computers. A new computer architecture based on the unique characteristics of tse logic is required.

The goal of this research is to contribute to the development of tse computer architectures through the design of tse logic circuits which perform the Golay transform skeletonizing algorithm. These tse

logic units will be controlled by conventional logic control units. A two-dimensional control unit would potentially allow dissimilar operations to be performed on various areas of an image simultaneously and independently. This flexibility, however, is not essential and can only be obtained with a significant increase in the number of tse logic components. Also, before the development of a two-dimensional control unit is inaugurated, the design of tse arithmetic-logic units should be thoroughly understood.

This chapter introduces some basic tse logic concepts and presents one tse logic implementation of the skeletonizing algorithm. Additions to the tse logic family proposed by Schaefer and Strong [1] are described. Several circuits are presented for generating Golay neighbor planes which simplify index recognition using tse logic.

Elementary Tse Processor Control

Control of basic tse processing units can be achieved by providing control images that consist of all ones when true and all zeros when not true. The control image can be created by switching a light source that illuminates each element in a fiber optic bundle. One possible light source is an array of electroluminescent devices manufactured on a semiconductor substrate. This source is similar to the output array of a standard active tse logic device and is assumed to be switched by a single CMOS compatible control line. The state of the output tse corresponds to the state of the control line.

Some tse processing requires control of individual data points within the tse. A number of different techniques can be used to set particular data points within a tse to the logic one or logic zero

level. One method of obtaining this control is to AND or OR the tse with a tse mask which contains a predetermined binary image. The tse mask image can be permanently stored in an electro-optical tse memory chip which contains an array of electroluminescent devices. A standard electroluminescent array based on the light source described earlier could be programmed to produce any required tse mask. Typically, the array would be programmed by specifying the final metalization pattern used in the manufacturing process to define the power connections to elements of the array. The same general technique is currently employed in the production of semiconductor read-only-memories. Alternately, the tse mask can be produced from an all logic one electroluminescent array by placing an "opaque and clear" photographic film mask between the array and the fiber optic image path. In cases where the only operation required is to set particular points within the data tse to zero, a film mask can be placed directly in the data path. Still another possibility is to design the active tse logic devices so that selected points within their output electroluminescent arrays can be permanently set to either logic level during manufacture. This might be accomplished by altering the final metalization pattern as described above or by forming the final integrated circuit interconnections using the recently developed dye laser micro-welder [16].

In both conventional and tse computer architectures, conditional operations are often performed when a zero state is detected. Signals to control these operations must be derived from a device which is capable of detecting an empty or all zero tse. The contractor and spiller devices described by Schaefer and Strong [1] can be combined to perform this function. The practical logic circuit implementations of

these devices produce long propagation delays which could seriously limit the throughput of a tse circuit. Therefore, a new tse device which produces an output image that is all ones if and only if the input image contains at least one logic one point should be developed. This device is referred to as a total spiller. A primary objective in the development of a total spiller should be to minimize propagation delay through the circuit.

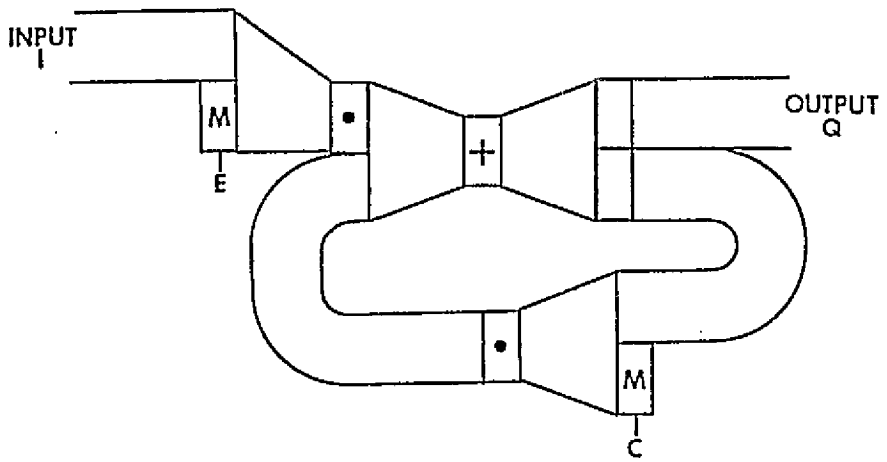
The timing relationships between tse control signals will depend upon the propagation delay introduced by each tse logic device in the circuit. At this time these propagation delays are not well defined; however, the response time of the electroluminescent material in active tse logic devices is expected to be the controlling factor. For this reason, the response times for the various active devices are likely to be approximately equal. Since some knowledge of propagation delays is required for efficient circuit design, all active tse logic devices except the spiller, combiner, and total spiller are assumed to introduce a maximum of one unit propagation delay. Because the spiller, combiner, and total spiller operations are more complex than the operation of a typical gate, a propagation delay of two units has been arbitrarily assigned to these devices. Images will be carried through tse buses and passive devices, such as sliders and interleavers, at the speed of light, so zero propagation delay is assumed for passive devices. These assumptions allow relative comparisons of the performance of tse logic designs and permit the design of conventional logic control units for tse circuits. Although control unit timing should be modified as the characteristics of real tse devices become known, the control techniques illustrated in these designs will be useful in the development of future control units.

Tse Memories

In the initial stages of tse logic development, integrated read-and-write tse memories are not expected to be available because of their complexity. Tse memory requirements will be met by using the integrated circuit one tse read-only-memories described in the previous section and by constructing read-and-write tse memories from standard tse devices. The simplest read-and-write tse memory is the OR latch illustrated in Figure 4.1. This device stores one binary image in a feedback path which is controlled by CMOS compatible signal C. The input path is controlled by a second CMOS compatible signal labeled E. Normally, the tse ROMs switched by C and E would contain all ones; however, special image patterns could also be used*** or the control input images could be generated by another tse circuit. When E = 1 and C = 1, the output image, Q^+, is Q + I (Q OR I). This property of the OR latch can be used to design logic circuits in which partial operations are performed sequentially and ORed into the tse latch to produce the final result.

A comparable tse latch based on the AND gate is shown in Figure 4.2. The input and feedback paths are controlled by CMOS compatible signals H and S, respectively. As specified by the control table, the output of the AND latch is Q · I (Q AND I) when H and S are both zero. This property of the AND latch can be used to form the final result of a tse operation by logically ANDing the results of a set of sequential operations. Although an objective of tse logic design is to attain

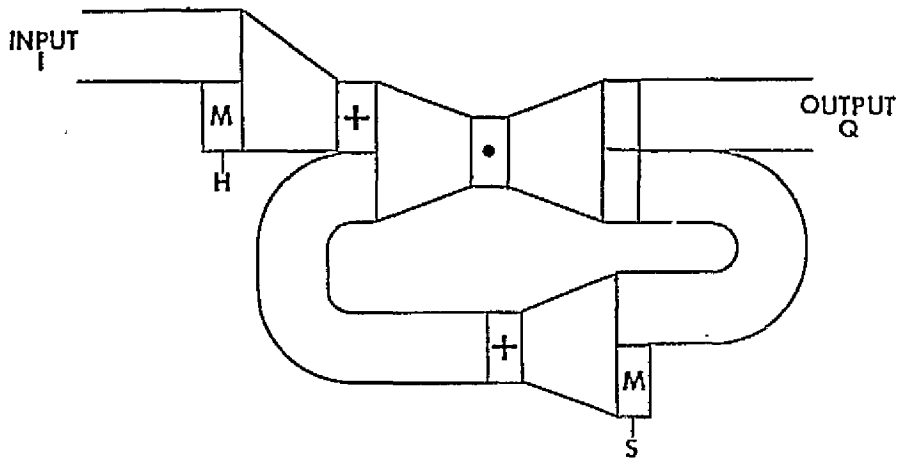
***The images stored in each tse read-only-memory used in this dissertation are defined in Appendix B.



CONTROL TABLE

E	C	Q^{+}_{ij}
0	0	0
0	1	Q_{ij}
1	0	I_{ij}
1	1	$Q_{ij}+I_{ij}$

Figure 4.1 One tse OR latch



CONTROL TABLE

H	S	Q_{ij}^{+}
0	0	$Q_{ij} \cdot I_{ij}$
0	1	I_{ij}
1	0	Q_{ij}
1	1	1

Figure 4.2 One tse AND latch

high speed operation through parallelism, serial logic circuits are often necessary for component minimization. Throughput can remain high because of the parallel nature of the tse logic devices themselves.

A major disadvantage of the basic latch circuits is that the output image is destroyed before storage of the input image can be guaranteed. In circuits where the next input to the latch depends upon the present output of the latch, incorrect operations can result. A master-slave memory, such as the one shown in Figure 4.3, can be used to avoid this difficulty. The timing diagram in Figure 4.3 illustrates the control signal sequence for storing a new tse in the master-slave memory. Timing is based on the following conservative, worst-case assumptions about the active tse logic devices:

Minimum propagation delay - 0 units
Maximum propagation delay - 1 unit
Minimum turn-on time - 0 units
Maximum turn-off time - 1 unit

The assumption of a minimum propagation delay of zero assures that correctly generated control signals will not permit race conditions to develop in tse circuits. In tse timing diagrams, the turn-on and turn-off timing constraints are illustrated by showing control signal state changes as if they occur over a time span of one gate delay. Actually, control signal state changes will occur instantaneously, and the state of the tse logic device being controlled can change at any time up to one gate delay later.

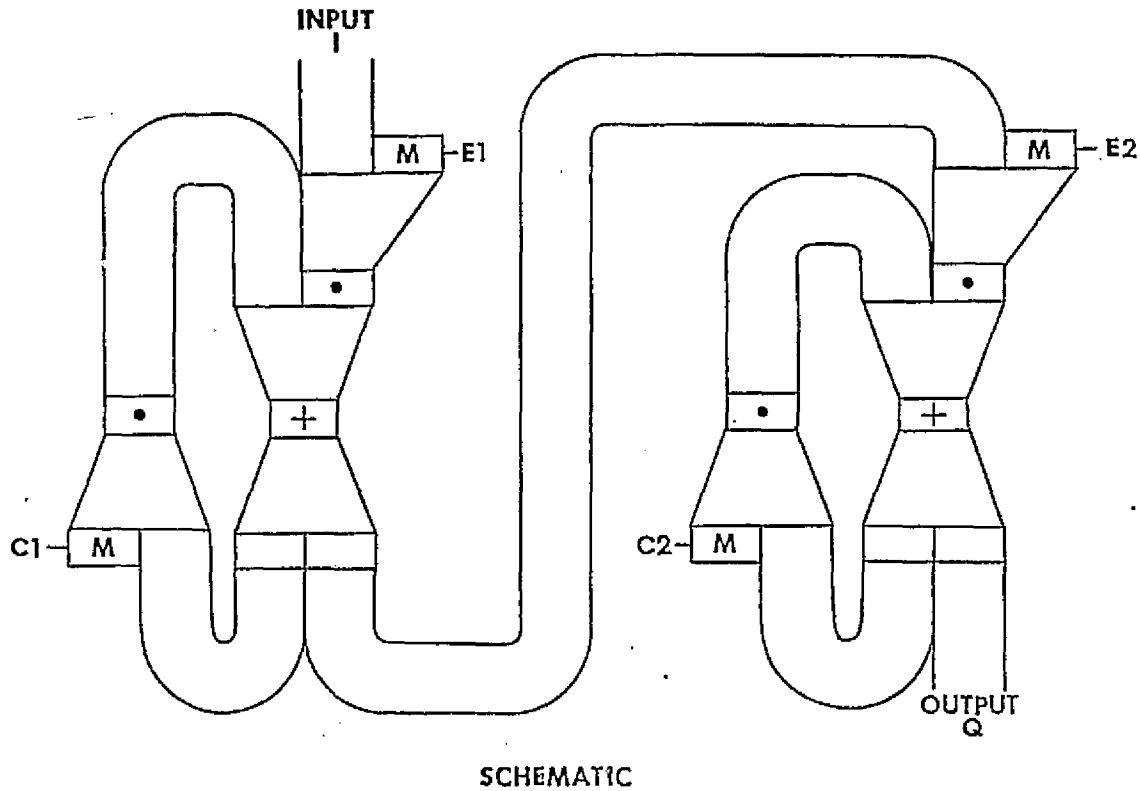
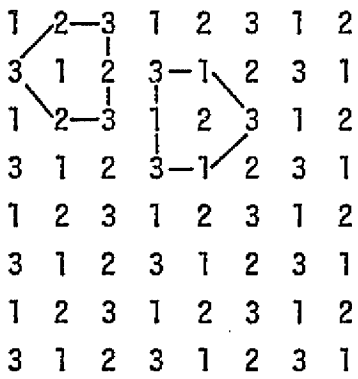


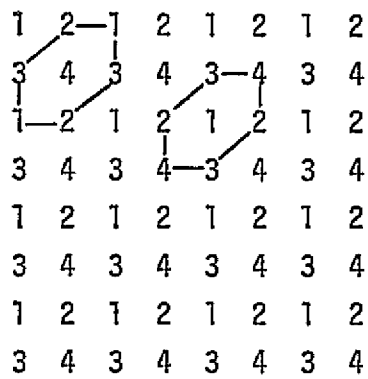
Figure 4.3 Master-slave tse memory.

Hexagonal-To-Rectangular Array Transformation

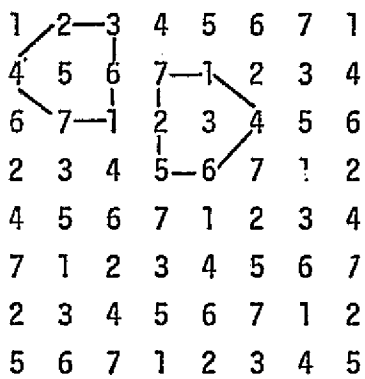
Standard tse logic devices utilize a rectangular array of binary data points and are not directly compatible with the hexagonal array employed in the Golay transforms. This difficulty can be overcome by noting that a rectangular array will be created from the Golay hexagonal digitization pattern if the data points in even numbered rows are shifted one-half unit distance to the left. The Golay neighborhood of each basis point, P, in the resulting rectangular array can be defined using the knowledge of which points formed the neighborhood of P in the original array. Subfield assignments and typical neighborhood patterns for rectangular arrays with three, four, and seven subfields are shown in Figure 4.4. Note that the neighborhood pattern for corresponding points in arrays of three and seven subfields are identical. Also note that data which is originally acquired through a hexagonal digitization pattern can be processed using the square tse logic array without introducing error. Data acquired through a scanning process can be automatically stored in a square array. If the data acquisition process is parallel, a special fiber optic bundle could be designed to convert the hexagonal array into a square array. Alternately, one could utilize a standard rectangular digitization pattern and merely assign particular Golay neighbors to each basis point. For sampling intervals which are small compared to the critical dimensions of interest in the binary image, the minor errors introduced by this process should be acceptable.



(a) Three Subfields



(b) Four Subfields



(c) Seven Subfields

Figure 4.4 Subfield assignments and neighborhood patterns for rectangular arrays with three, four, and seven subfields.

Golay Neighbor Planes

Golay's modular operations are normally functions of the index of the basis point, P. As a result, a Golay transform processor must be capable of performing logical operations on a point P_{ij} as a function of its coplanar Golay neighbors which, in the case of four subfields for example, are located at positions $P_{i,j-1}$, $P_{i-1,j}$, $P_{i-1,j+1}$, $P_{i,j+1}$, $P_{i+j,j}$, and $P_{i+1,j-1}$. In constrast, binary tse logic operations are performed on separate image planes in parallel, and the state of a point X_{ij} in the output array is only a function of the state of corresponding points in position Y_{ij} of the input image arrays. Thus, for index recognition using tse logic, the state of each neighbor of a basis point, P_{ij}, should be available in position Y_{ij} of a separate image plane. Six Golay neighbor planes (GA, GB, GC, GD, GE, and GF) are required to represent the state of each neighbor of every basis point in a tse.

A tse logic circuit which will produce the six Golay neighbor planes for a tse divided into four subfields is shown in Figure 4.5. Plane GA is formed by sliding the input image right one element. This action transfers a point in position $P_{i,j-1}$ of the original image to position P_{ij} of plane GA. Therefore, each point in plane GA is the A Golay neighbor of the point which occupies the same relative location in the original tse. Similarly, plane GB contains the B Golay neighbors, and so forth. The input image is assumed to have a one element deep border of dummy zeros since a complete neighborhood cannot be defined for the border elements.

When a tse is divided into three or seven subfields, a somewhat more complex Golay neighbor planes generator circuit is required because

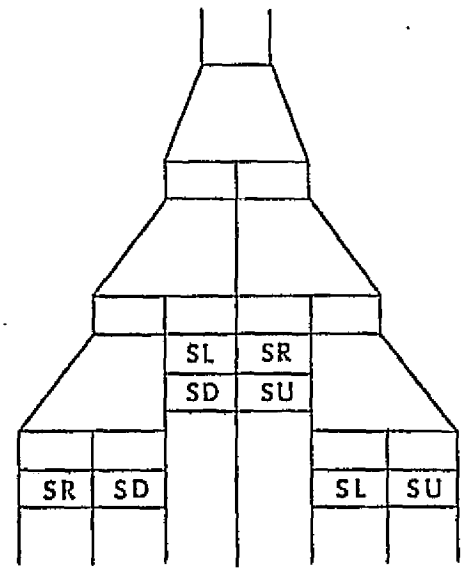


Figure 4.5 Golay neighbor planes generator for rectangular arrays with four subfields.

the surrounds of basis points in even and odd numbered rows are oriented in opposite directions. The A, B, C, D, E, and F Golay neighbors of basis point P_{ij} are $P_{i,j-1}$, $P_{i-1,j}$, $P_{i-1,j+1}$, $P_{i,j+1}$, and $P_{i+1,j+1}$ respectively, if i is even but are $P_{i,j-1}$, $P_{i-1,j-1}$, $P_{i-1,j}$, $P_{i,j+1}$, $P_{i+1,j}$, and $P_{i+1,j-1}$, respectively, if i is odd. Since $P_{i,j-1}$ is neighbor A in each case, plane GA can be created by sliding the original tse right one element. However, sliding the input tse down one element in an attempt to create plane GB actually results in an image which contains B Golay neighbors in even rows and C Golay neighbors in odd rows. To obtain a tse with B Golay neighbors in odd rows, one can slide the original image down one element and to the right one element. By combining the odd rows of this tse with the even rows of the previous image, a single tse which corresponds to Golay neighbor plane GB can be obtained. This process is illustrated in Figure 4.6 which is a schematic for one possible Golay neighbor planes generator circuit for images divided into three or seven subfields. As shown in Appendix B, the one tse read-only-memory labeled ME consists of all ones in odd rows and all zeros in even rows. Mask MO is the complement of ME. The Type 1 Golay neighbor planes generator circuit requires 38 active and 29 passive tse logic devices. Based on the standard assumption of one unit gate delay for each active tse logic device, the complete circuit has a propagation delay of six gate delays.

Figure 4.7 illustrates the advantages which can be obtained by replacing active tse read-only-memory masks with film masks inserted directly in the signal path. The boxes labeled FE and FO represent photographic film masks which contain the same patterns as the active tse masks ME and MO shown in Figure 4.6. An opaque area of the film

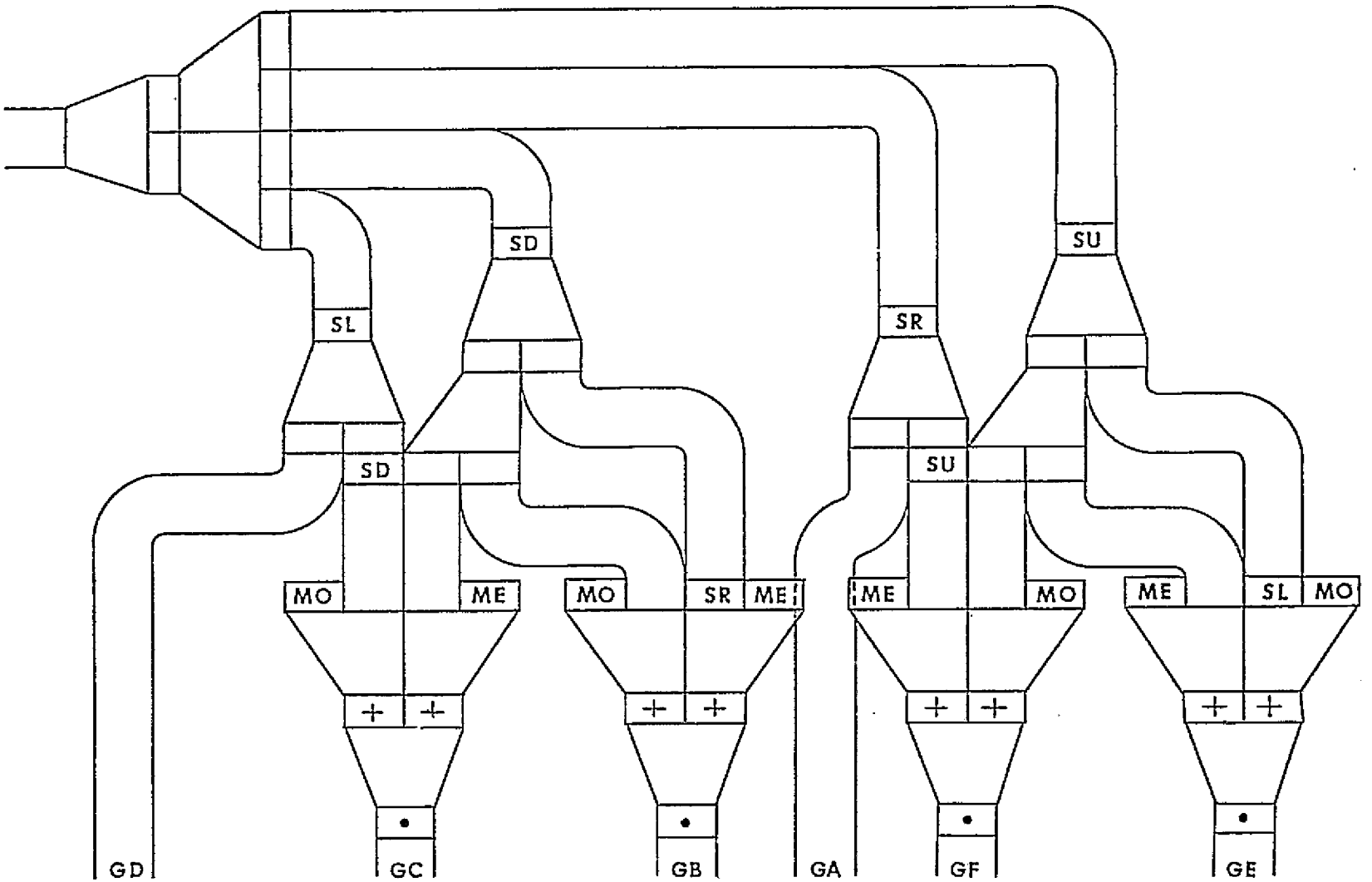


Figure 4.6 Type 1 Golay neighbor planes generator for rectangular arrays with three or seven subfields.

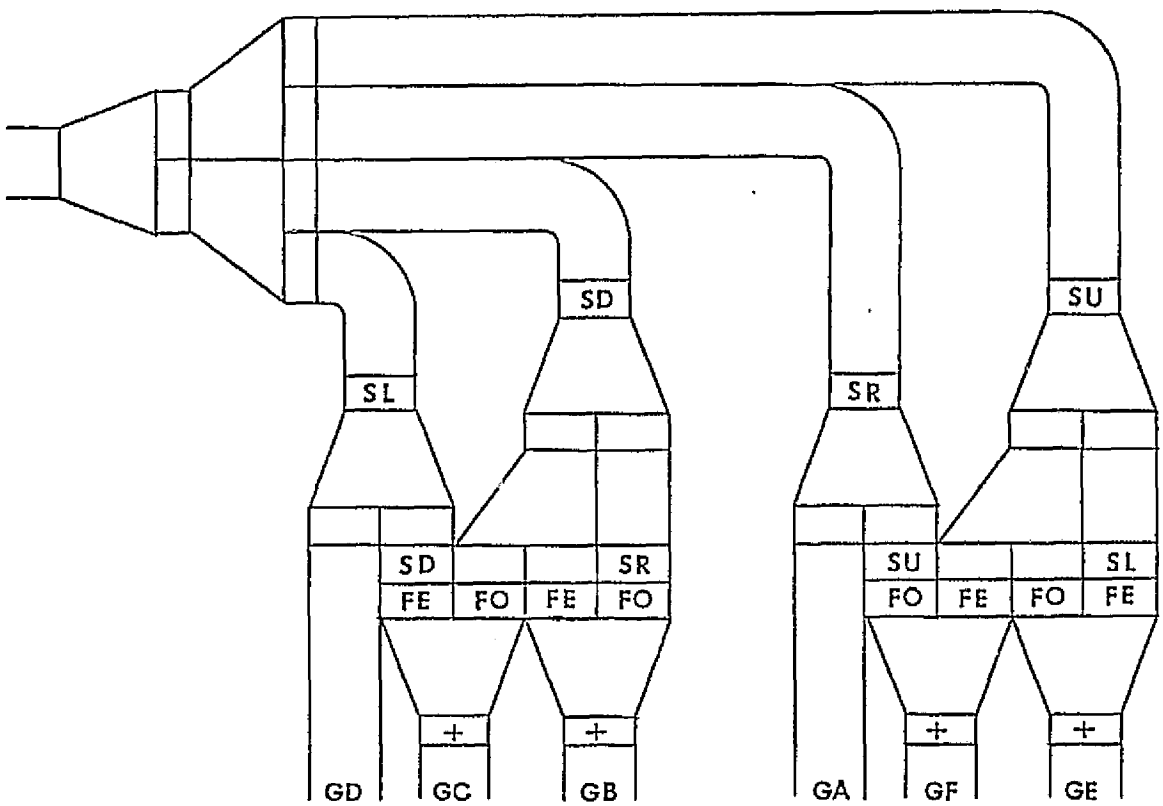


Figure 4.7 Type 2 Golay neighbor planes generator for rectangular arrays with three or seven subfields.

forces the corresponding tse data point to zero by blocking the light. Clear areas of the film have no effect on their corresponding tse data points. This implementation of the Golay neighbor planes generator circuit requires 22 active and 21 passive tse devices. Circuit propagation delay is five gate delays, one less than for the Type 1 circuit. The technique of programming selected points in the output array of an active tse device to one or the other logic level could be used to implement the Type 2 Golay neighbor planes generator without photographic masks.

The Golay neighbor planes generator circuit can be simplified even further by using the proposed new tse logic device illustrated in Figure 4.8. An EXCHANGE gate is a passive tse intercommunication device with two tse inputs (A and B) and two tse outputs ($\tilde{A}$ and $\tilde{B}$). The EXCHANGE gate is constructed from optical fibers using the same techniques employed in the construction of tse interleavers and image buses. Output image path $\tilde{A}$ contains the fibers which carry the data from the odd rows of the image A input and the fibers which carry the data from the even rows of the image B input. The $\tilde{B}$ output path contains fibers from the even rows of A and the odd rows of B. Thus, the EXCHANGE gate performs the operation of exchanging the data in the alternate rows of two images. This is one of the operations required to generate the Golay neighbor planes for tses divided into three or seven subfields. A schematic for the equivalent circuit of the row EXCHANGE gate is presented in Figure 4.9. Since the tse array is square, a similar column exchange operation can be obtained by rotating the EXCHANGE gate fiber optic array 90° with respect to the image buses interfaced to the in-

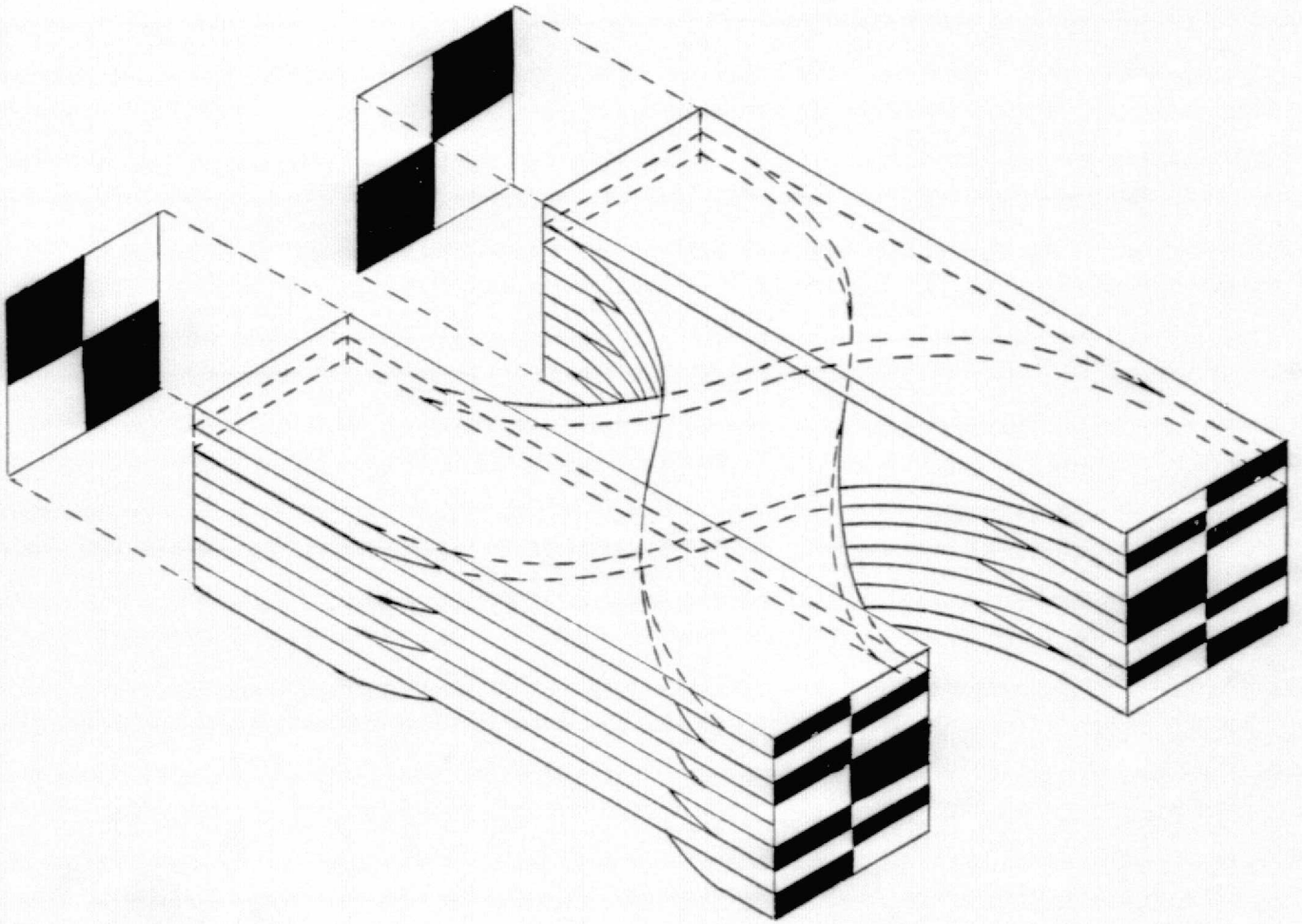


Figure 4.8 Tse EXCHANGE gate.

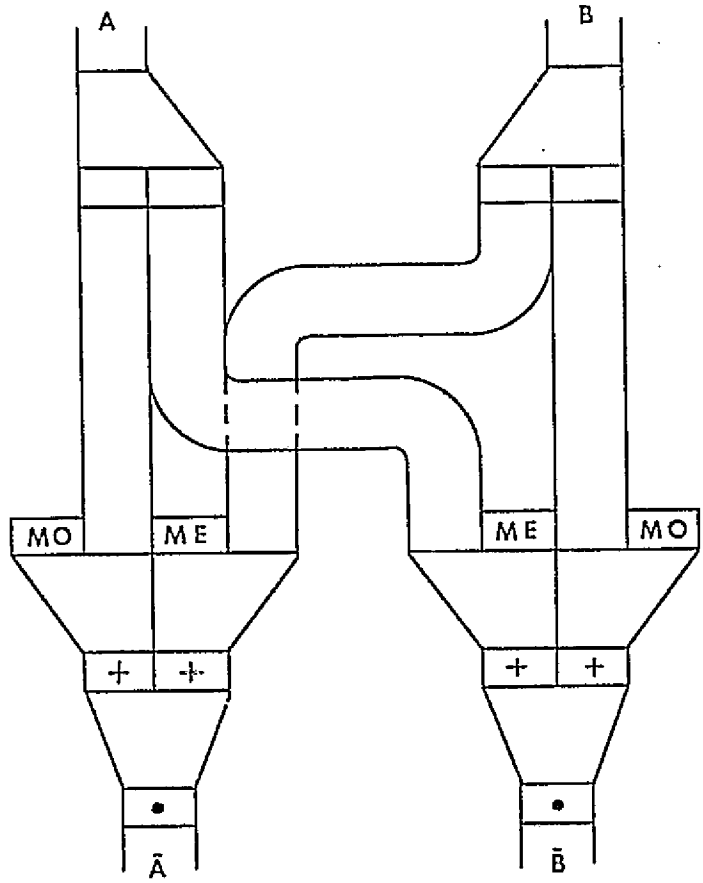


Figure 4.9 Equivalent circuit for an EXCHANGE gate.

puts and outputs of the device. The EXCHANGE gate illustrates that special fiber optic arrays can be constructed to perform certain tse logic functions that would normally require several active devices and tse masks. The special tse logic devices reduce circuit power consumption and propagation delay.

EXCHANGE gates are employed in the third type of Golay neighbor planes generator circuit shown in Figure 4.10. Only one output of each EXCHANGE gate is used. This implementation of the Golay neighbor planes generator requires only 18 active and 21 passive tse logic devices. Propagation delay is reduced to four gate delays by the passive nature of the EXCHANGE gates. A comparison of the three types of Golay neighbor planes generator circuits is provided in Table 4.1.

Index Recognition

With the neighborhood points for each element of an image available in the six Golay neighbor planes, the task of designing circuits to perform any binary function of a basis point and the basis surround reduces to a relatively standard logic design problem. Tse logic circuits can be designed to recognize any surround index or combination of indices. An index recognition circuit can be a sequential circuit or a pure combinational circuit. The major distinction between the conventional logic design procedure and the required tse logic design procedure is that many of the traditional minimization procedures do not apply to tse logic design because of the fan-in and fan-out restrictions. An important similarity between conventional and tse logic circuit designs is that a trade off generally exists between operating speed and circuit complexity. This fact is demonstrated by the index

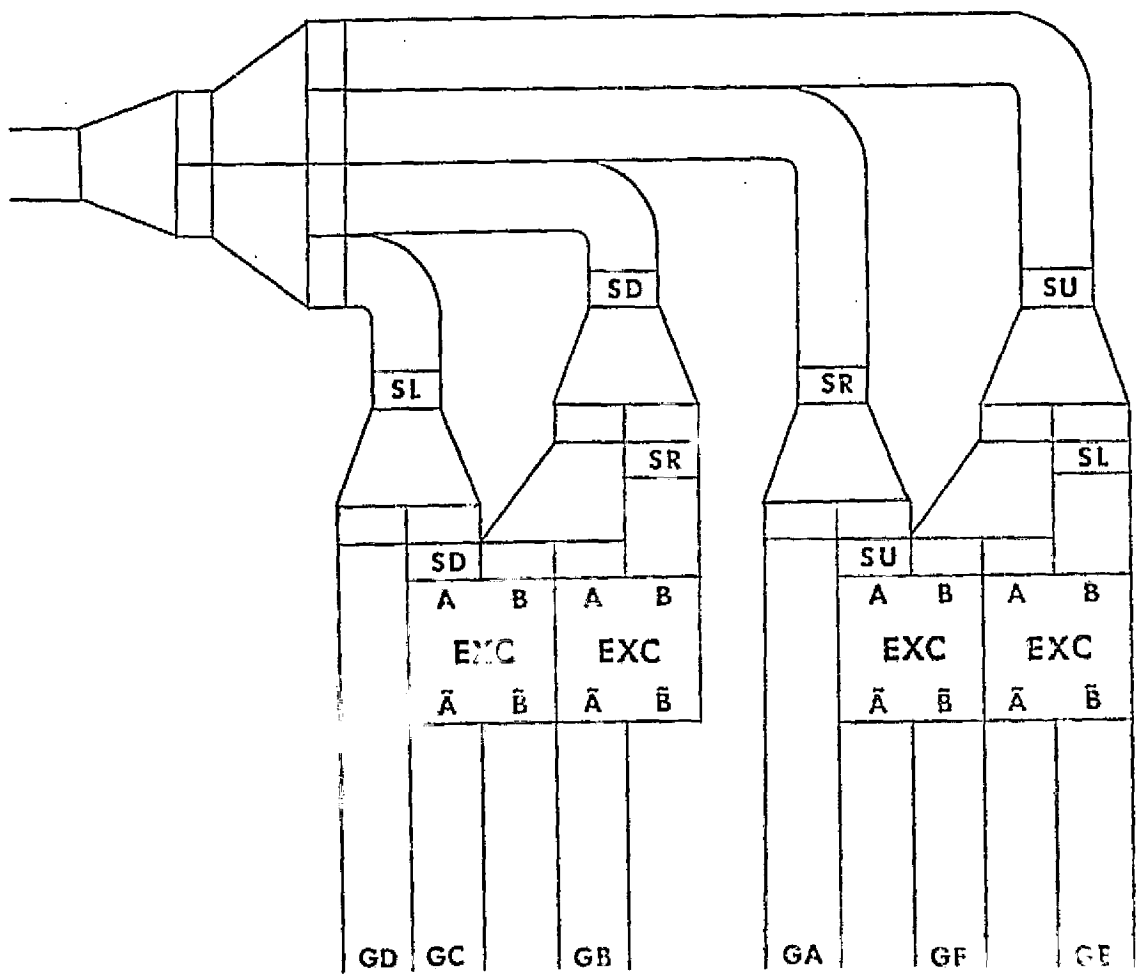


Figure 4.10 Type 3 Golay neighbor planes generator for rectangular arrays with three or seven subfields.

TABLE 4.1

CHARACTERISTICS OF THE GOLAY NEIGHBOR PLANES GENERATOR CIRCUITS FOR RECTANGULAR ARRAYS OF THREE OR SEVEN SUBFIELDS

Circuit Type	Figure Number	Number of Passive Devices	Number of Active Devices	Propagation Delay in Standard Gate Delays
Type 1	4.6	29	38	6
Type 2	4.7	21	22	5
Type 3	4.10	21	18	4

recognition circuits presented below.

The Golay transform skeletonizing algorithm requires recognition of basis points which have a surround index of one, two, or three. Figure 4.11 shows the schematic of a combinational circuit which produces an output tse with zeros marking the positions of basis points that have an index of one, two, or three. Each cell in the space iterative circuit recognizes one of the six rotationally equivalent orientations of a surround with index one, two, or three. Six cells are required to recognize all of the possible orientations of the surrounds. This index recognition circuit has a propagation delay of only ten gate delays but requires 125 active and 77 passive tse logic devices.

Figure 4.12 illustrates a time iterative index recognition circuit which determines the index by comparing the input tse to tse masks. Each Golay neighbor plane is EXCLUSIVE-ORed with a tse mask to produce a tse with logic one points marking positions where the neighbor plane and the mask have complementary values. These six planes are ANDed together to form the input to the tse latch. The tse latch input contains logic one points only in positions where none of the corresponding Golay neighbor plane points match their corresponding masks. If all of the input masks are turned on, the tse latch input image will contain logic one points in the positions of basis points with a surround of index zero. Each of the 64 possible combinations of input mask states correspond to one orientation of one of the 14 possible indices. All basis points with a particular surround index can be recognized by sequentially checking for every orientation of that surround and ORing the partial results into the tse latch. The timing diagram shown in

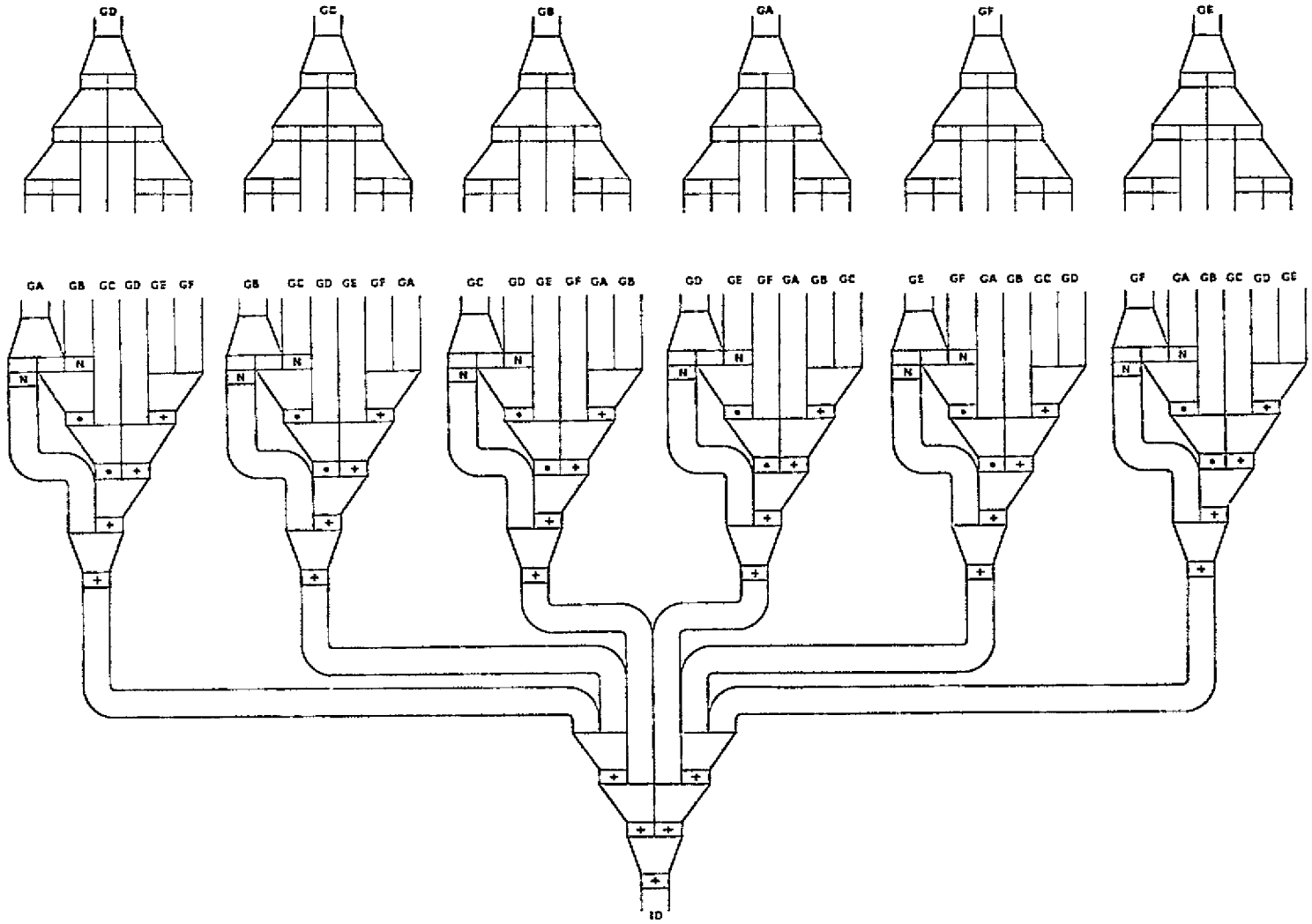


Figure 4.11 Space iterative combinational index recognition circuit for indices one, two, and three.

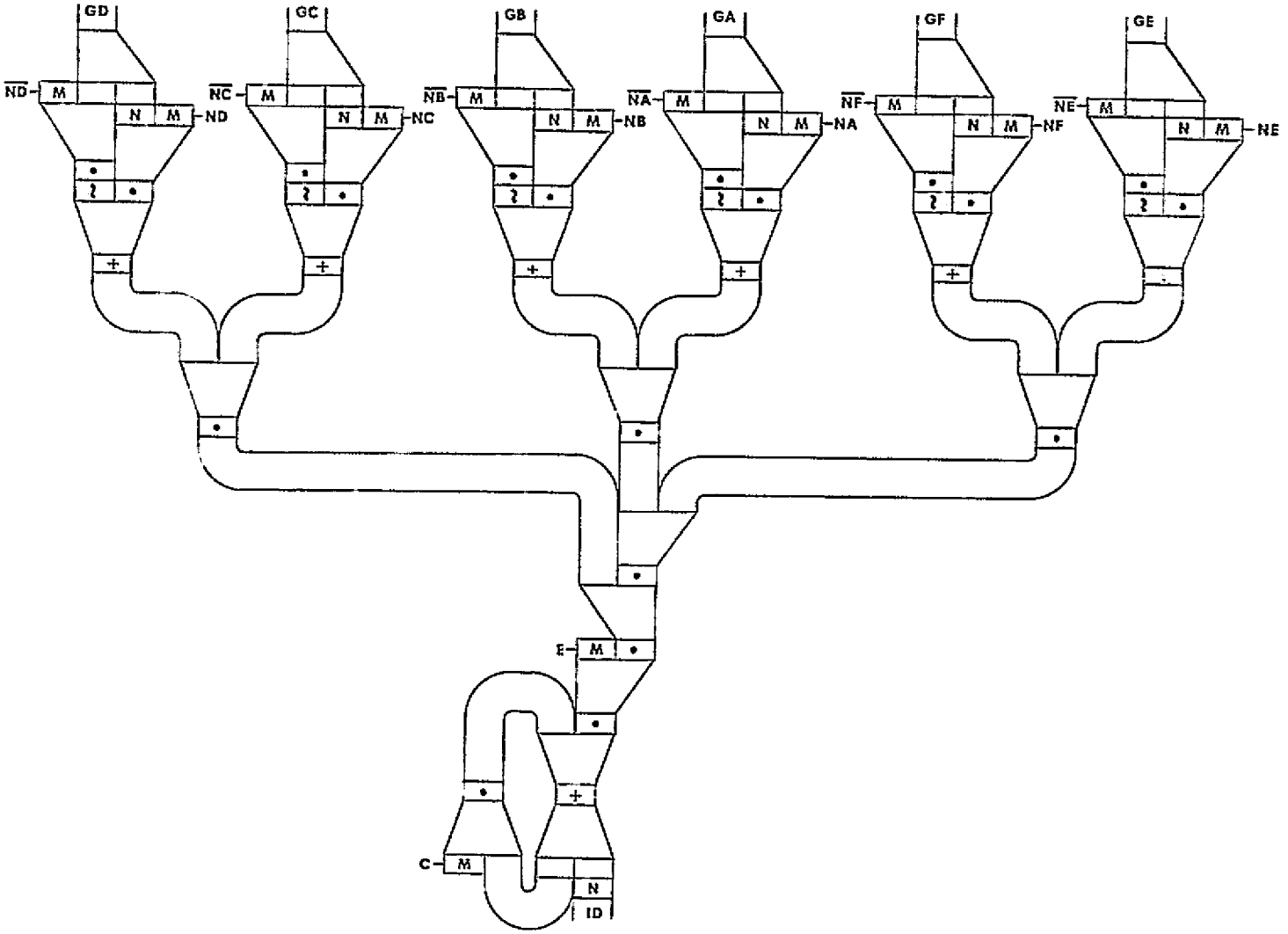


Figure 4.12 Time iterative, comparison type index recognition circuit.

Figure 4.13 illustrates the control sequence required to recognize an index with a weight of three. To perform the skeletonizing algorithm, basis points with index one, two, or three must be recognized. This procedure takes 215 unit gate delays but requires only 61 active and 33 passive tse logic devices (Table 4.2). An important feature of this index recognition circuit is that any index or combination of indices can be recognized. With the addition of a NEGATE operation, recognizing that the index of a basis point is not an element of a specified set A is equivalent to identifying the index as an element of set B where the union of A and B is the set of all 14 indices. The upper bound on the time required to recognize all basis points with any given set of indices is established by the time required to identify half of the 64 possible surround orientations. Thus, in the worst case, 383 unit gate delays are required to identify all basis points with indices from a particular set.

Golay Function

Once the index recognition circuits have been designed, a Golay function circuit can be developed to perform the logical operations specified by a particular Golay transform. The Golay function circuit for performing the swelling operation [10] defined by the symbol

$$M^{3}_{3\text{-}5,\infty}[k = a_{i(k)} + \overline{a_{i(k)}} \cdot k] \qquad (5)$$

is illustrated in Figure 4.14. As shown in Figure 4.15, the skeletonizing algorithm requires an even simpler Golay function circuit. Similar Golay function blocks can be designed for any Golay transform.

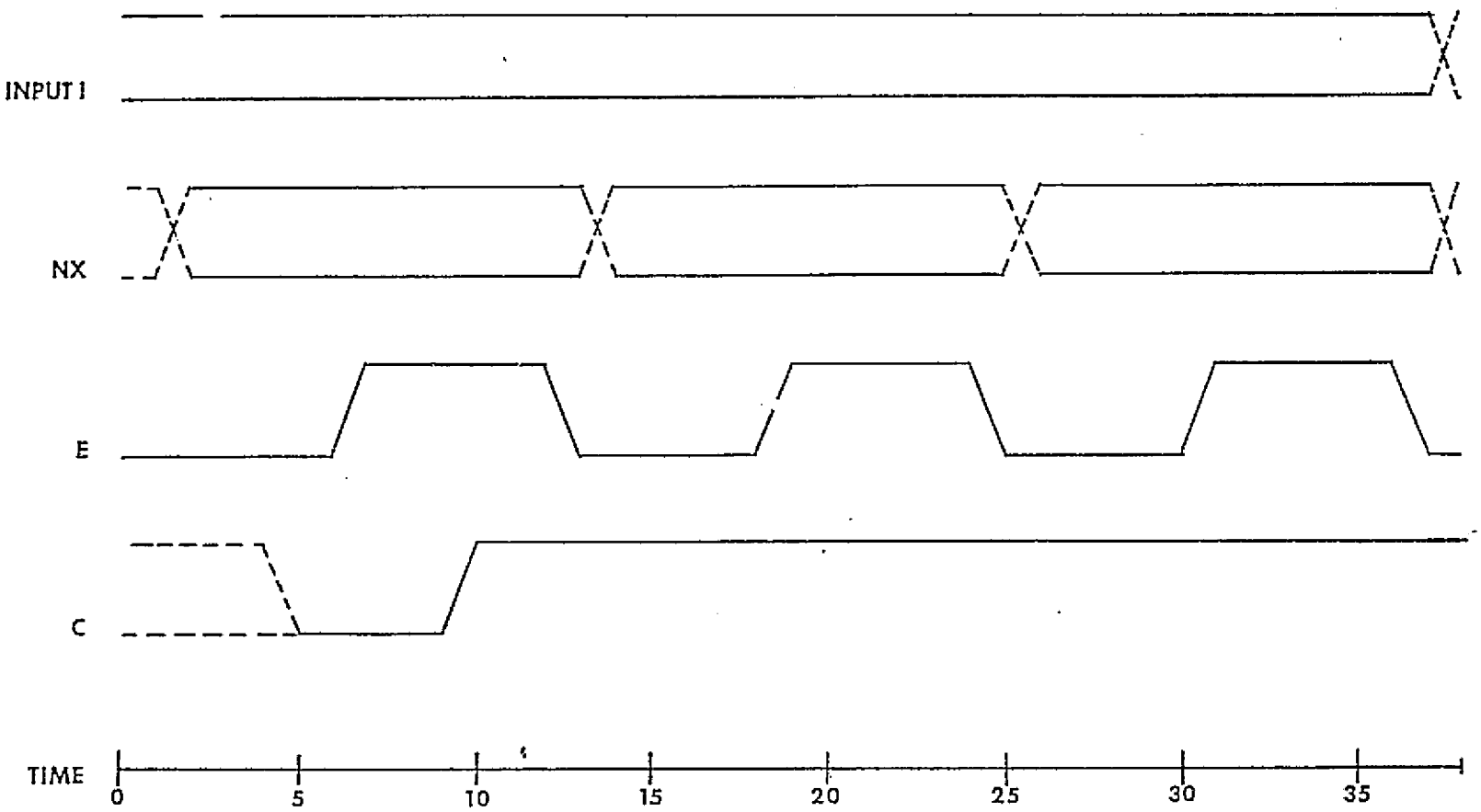


Figure 4.13 Timing diagram for recognizing an index with a weight of three using the comparison type index recognition circuit.

TABLE 4.2

CHARACTERISTICS OF THE SPACE AND TIME ITERATIVE
INDEX RECOGNITION CIRCUITS

Circuit Type	Figure Number	Number of Active Devices	Number of Passive Devices	Time Required to Recognize Indices 1, 2, and 3 (Unit Gate Delays)
Space Iterative	4.11	125	77	10
Time Iterative	4.12	61	33	215

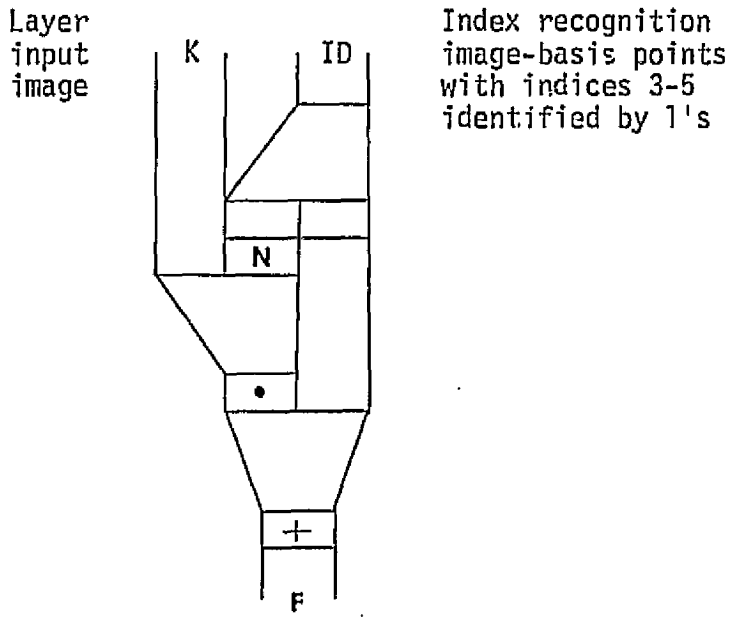


Figure 4.14 Golay function circuit for a swelling operation.

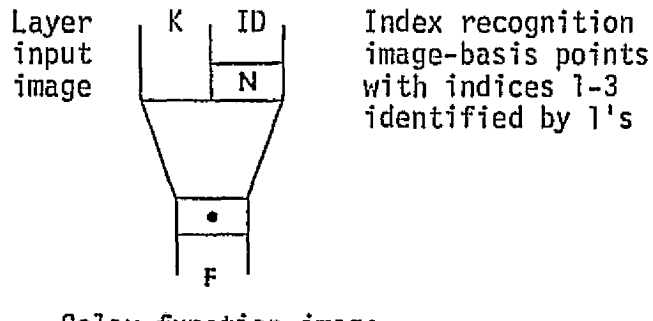


Figure 4.15 Golay function circuit for a skeletonizing operation.

Note that although the Golay function is performed simultaneously on all points of the layer input image, the results at any one time are only valid for one subfield within the image. The Golay function must normally be performed sequentially on each subfield of the image. A hardwired tse logic circuit which illustrates the procedure for performing a Golay transform is presented in the next section.

Implementation of the Skeletonizing Algorithm

The task involved in the Golay transform skeletonizing algorithm,

$$M^{3}_{1-3,}\ [k = a_{i(k)} \cdot k], \tag{6}$$

is to shrink all simple blobs in the binary image plane, K, to a single one, while reducing blobs with holes to one or more loops consisting of a single layer of ones. During an iteration through the algorithm, the specified modular operation (Golay function) is performed in parallel on the points within each of the three subfields in sequence. As each subfield is processed, the basis points within that subfield which are in state one and have a surround index not equal to one, two, or three are allowed to remain in state one while all other points in that subfield are set to zero. The process is complete when further iterations produce no change in the output image.

A block diagram of the tse logic implementation of the skeletonizing algorithm is presented in Figure 4.16, and a schematic for the circuit is illustrated in Figure 4.17. This particular design is intended for high speed processing and, therefore, utilizes the space iterative, combinational index recognition circuit. The timing diagram

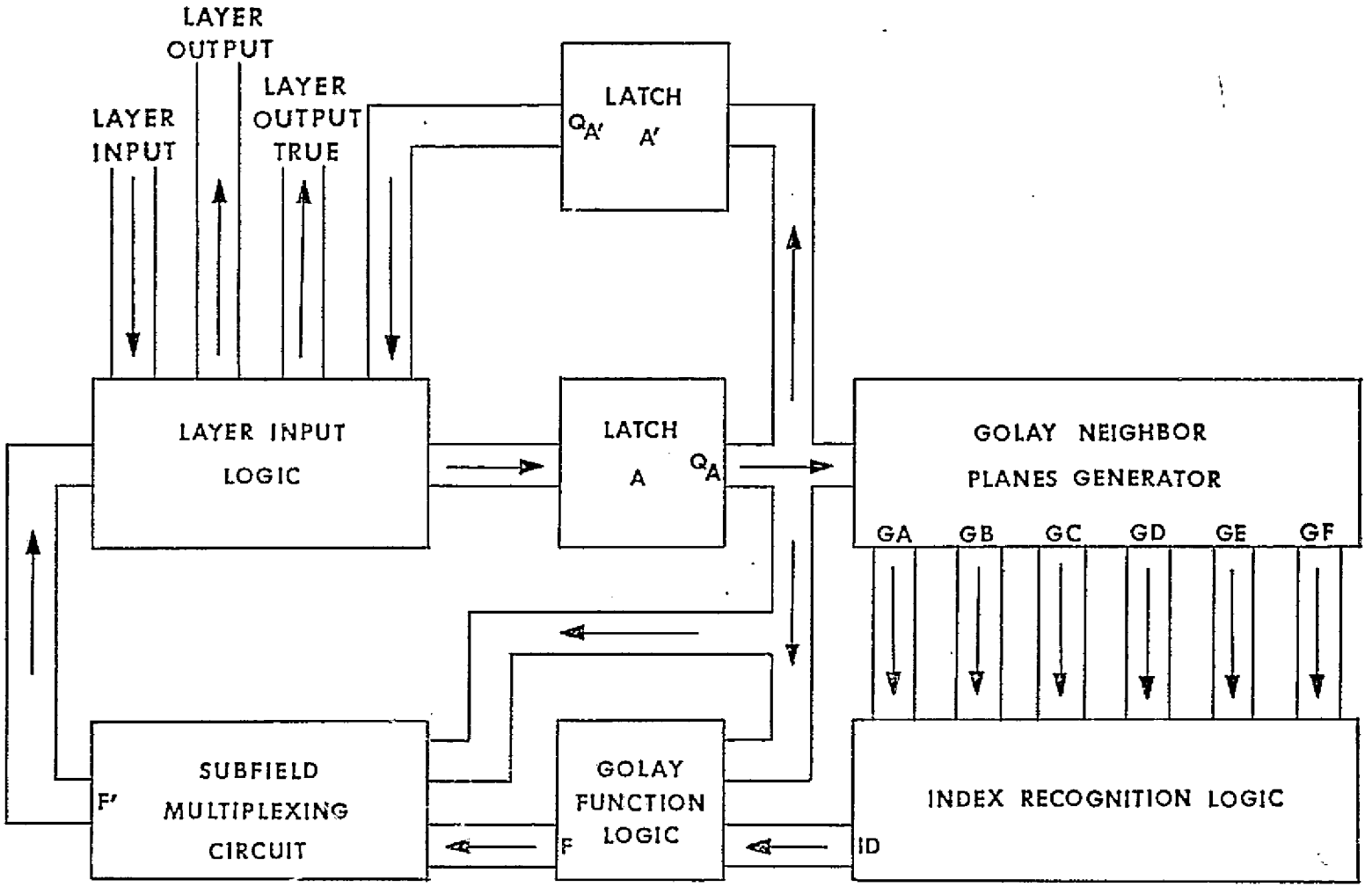


Figure 4.16 Block diagram of a tse logic implementation of the skeletonizing algorithm.

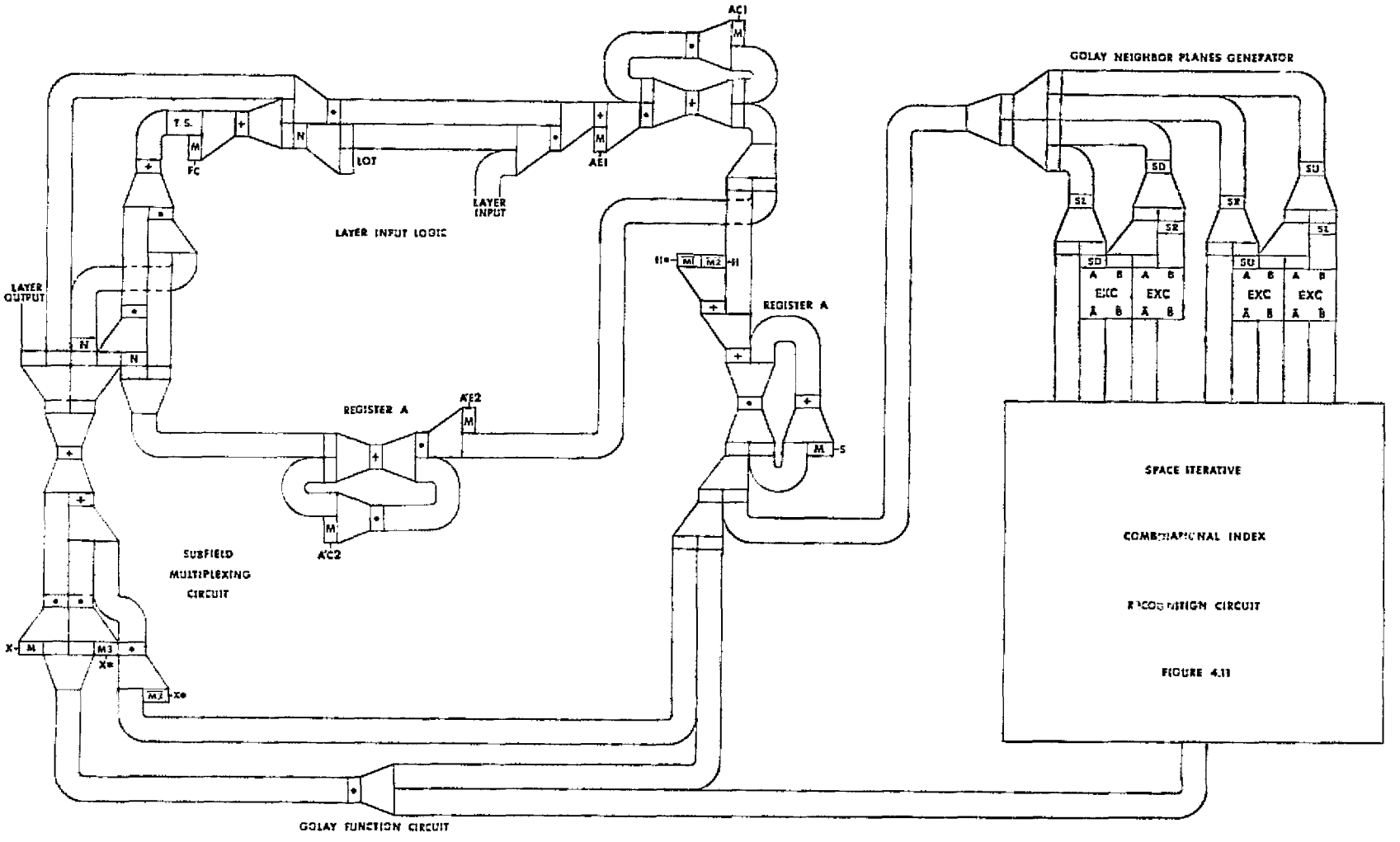


FIGURE 4.11

Figure 4.17 Schematic of a hardwired skeletonizing machine.

provided in Figure 4.18 shows the sequence of control signals required by the skeletonizing machine.

The operation of the skeletonizing machine can be explained by following an image, K, through one iteration of the skeletonizing algorithm. Assume that the binary image labeled K in Figure 4.19 is available at the layer input of the skeletonizing machine. Upon completion of the current task, the layer input logic will automatically gate (t = 0) the new image plane K into latches A and A´ for temporary storage. Golay neighbor planes are generated using EXCHANGE gates (t = 18), and basis points with a surround whose index is not one, two, or three are identified by the index recognition circuit. At t = 28 the output of the index recognition circuit is ANDed with image K to obtain the Golay function output image, F(t = 29). Although the entire image is being processed, only the first subfield results are valid. Image F propagates unaltered through the subfield multiplexing circuit and the layer input logic to the input of latch A. Subfield one of image F´ is then ANDed into latch A(t = 48) by manipulating the tse mask control lines so that only the first subfield portion of the register A input control image changes. Latch A now contains subfields two and three of the original image, K, and a new subfield on which the Golay modular operation has been performed. The first subfield operation is now complete. The image created by the first subfield operation is allowed to propagate back through the circuits described above with the result that the second subfield is processed and ANDed back into latch A(t = 82). After the completion of the second subfield operation, the resultant binary image is again processed by the Golay neighbor planes generator (t = 86), the index recognition circuit

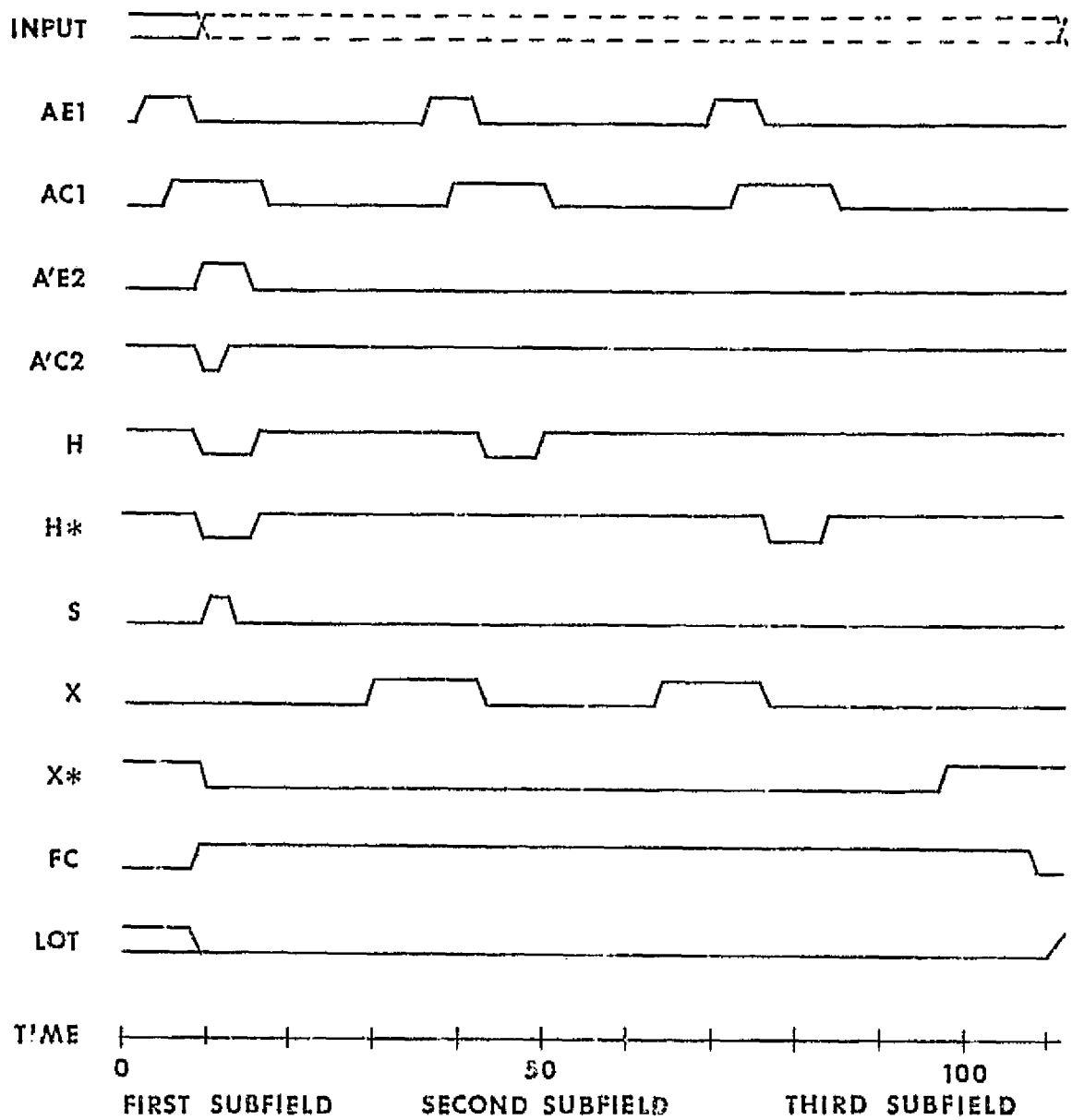


Figure 4.18 Timing diagram for one iteration of the skeletonizing algorithm.

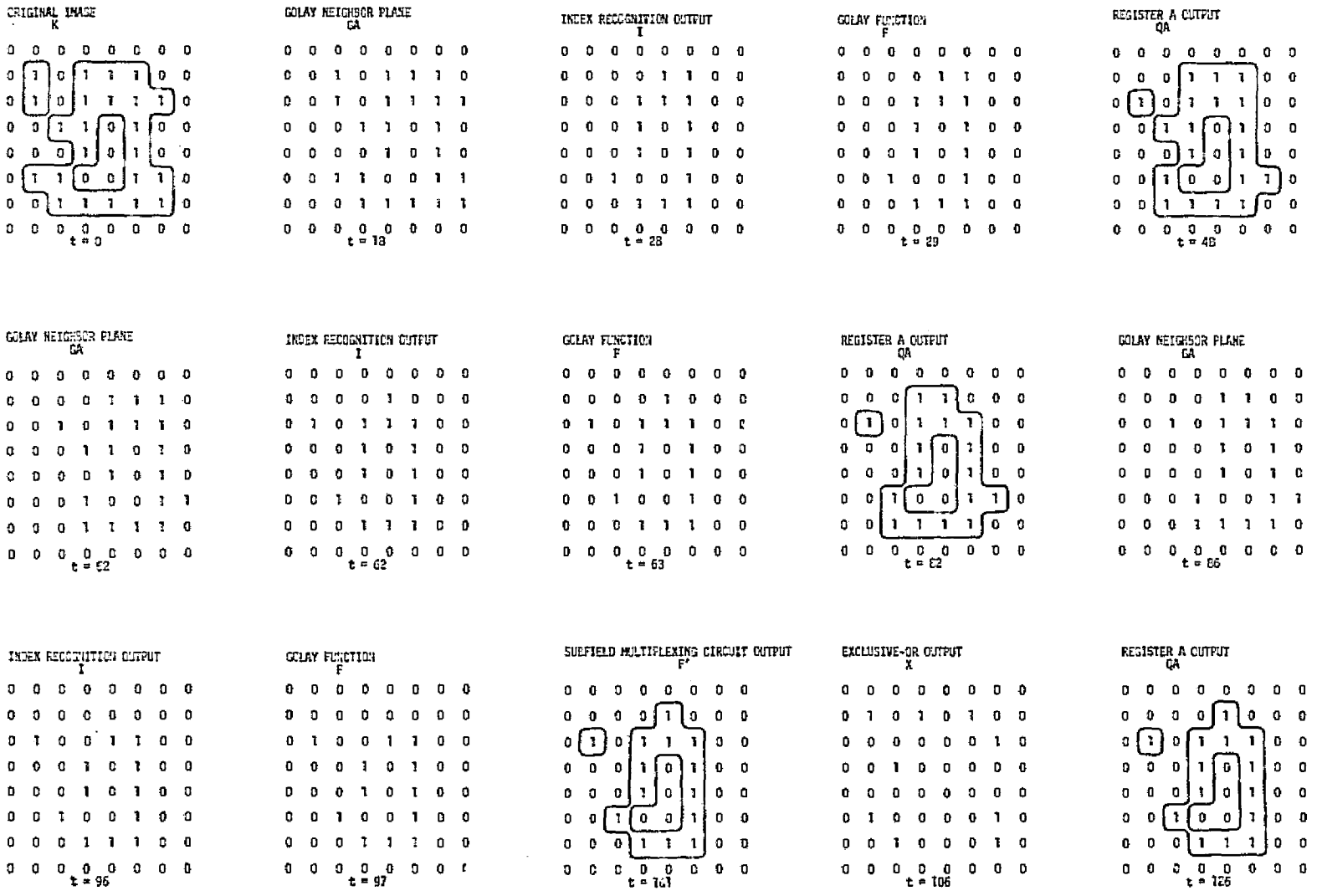


Figure 4.19 Images formed during one iteration of the skeletonizing algorithm.

(t = 96), and the Golay function circuit (t = 97). Since the third subfield operation marks the completion of one iteration, the resultant subfield is not ANDed directly back into latch A. Instead, the third subfield of image F is combined with subfields one and two from image latch A by the subfield multiplexing circuit to produce tse F′(t = 101). Image F′ is EXCLUSIVE-ORed (t = 106) with the original image K which has been stored in latch A′. The resultant image contains a one in the position of any basis point which changed state as a result of a modular operation performed during the current iteration. If no points have been altered, image F′ is the Golay skeleton of the original image K and processing is complete. The total spiller produces a control tse which allows a new image, labeled J, to be accepted for processing and causes the layer output true image to become all ones. However, if any basis point was altered during the current iteration, another iteration is required. In that case, the control image produced by the total spiller allows image F′ to be stored in latches A and A′(t = 126). Processing then continues as described above until the skeleton of the image is obtained.

Using this skeletonizing machine, each iteration of the skeletonizing algorithm takes 112 unit gate delays. The total image processing time is, of course, dependent upon the number of iterations required to obtain the skeleton of a particular image. Processing time is not a function of the size of the tse array. A total of 208 active and 134 passive tse components are required for this implementation of the skeletonizing operation. Of these, 116 active and 58 passive components are required because of the fan-in and fan-out limitations of

the electro-optical tse logic devices. The number of basic tse logic components required to perform the skeletonizing algorithm can be reduced to 146 active and 91 passive devices by using the comparison type index recognition circuit and a negator. The time for each iteration is increased to 679 unit gate delays.

Evaluation of Skeletonizing Machines

Adequate evaluation and comparison of skeletonizing machines requires a set of performance parameters. Several performance characteristics of the skeletonizing machine described in this chapter are given in Table 4.3. A hardware cost function defined by

$$\text{Hardware Cost} = Ax + By + Cz$$

where x = number of active tse devices

y = number of passive tse devices

z = number of unit lengths of fiber optic bundle

and the constants A, B, and C are weighting factors which are determined from the actual size and price of the components

is proposed for comparison of various machines. This cost function provides a more complete representation of circuit size and power consumption than a total component count. The distinction between active and passive tse devices is maintained because active devices consume power without contributing significant weight or bulk, whereas the passive devices consume no power but contribute substantially to circuit weight and bulk. The interconnecting fiber bundles are also important because of their weight and bulk. The length of fiber bundles

TABLE 4.3

BASIC HARDWIRED SKELETONIZING MACHINE PERFORMANCE CHARACTERISTICS

Cost Function	Number of Control Signals	Total Component Count	Gate Delays per Iteration (Time in Seconds)	Data Rate Simple Images per Minute	Average Power Consumption in Watts	Peak Power Consumption in Watts	Speed-Power Product in Watt-Seconds
208A+134B	10	342	112 (0.56)	107.14	606	609	339.36

required for a particular circuit cannot be accurately determined at this time, and, therefore, is not included in Table 4.3.

The relative speed of various skeletonizing machines can be most accurately determined by considering the average rate at which simple images can be processed. A simple image is defined as any image whose skeleton is itself. Only one iteration is required to process a simple image. The simple image processing operation represents the most basic task which includes all operations required in the general skeletonizing algorithm. Note, however, that the time required to process a typical image will be much greater than the time required to process a simple image.

Both peak and average power consumption data are provided in Table 4.3. Peak power consumption provides an indication of the relative cost and size of the power supplies required for the circuit. Average power consumption is important for space applications and in battery powered equipment where the total available energy is limited. A generally accepted figure of merit which includes consideration of machine speed and power consumption is provided by the speed-power product. The speed-power product listed in Table 4.3 reveals the energy which must be expended to perform one skeletonizing operation. The time and power consumption figures given in Table 4.3 are based on the five millisecond propagation delay and three watt power consumption projected by NASA [1] for early prototype tse devices.

CHAPTER 5

IMPLEMENTATION OF THE SKELETONIZING ALGORITHM USING TSE LOGIC DEVICES WITH A DISABLE INPUT

The tse logic control technique described in Chapter 4 permits conventional logic control of tse circuit operations while requiring the CMOS compatible output disable signal to be provided for only one basic tse device, the electroluminescent array read-only-memory. Each control signal requires one passive and two active tse devices (Figure 5.1) to complete the interface with the tse circuit. One unit gate delay is also added to the tse data path. Two improved skeletonizing machines and a technique for reducing the number of tse devices required for the control signal interface are presented in this chapter. In addition, a conventional logic organization for the control units of dedicated tse logic circuits is developed, and several index recognition circuits which illustrate the trade off between circuit complexity and operating speed are characterized.

Improved Conventional Logic Control Signal Interface to Tse Circuits

A majority of the control signals applied to tse logic circuits are used to disable image propagation through selected data paths. This goal could also be accomplished by deactivating the output electroluminescent array of the active tse logic devices from which the image is originating. A CMOS compatible control line which activates the array when in

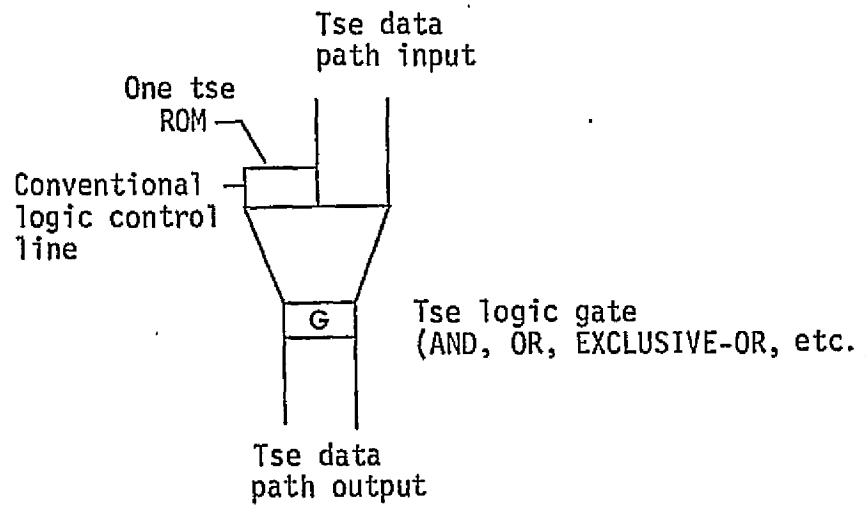
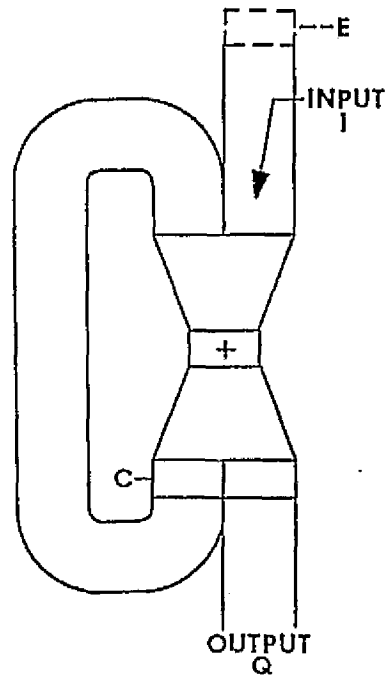


Figure 5.1 Basic conventional logic control signal interface to tse circuits.

the logic one state and deactivates the array when in the logic zero state can be provided for each active tse logic device. Figure 5.2 illustrates the hardware cost reduction obtained when a tse OR latch is constructed using this control technique. Note that control signal E is applied to the active tse device which creates the input image. The hardware cost function for the improved one tse OR latch is 3A + 2B compared to a hardware cost of 7A + 4B for the basic one tse OR latch (Figure 4.1, page 52). Circuit operating speed is also improved by this control technique. One disadvantage of the control technique is that the complexity of tne integrated circuit, active tse logic devices is increased. This disadvantage should be offset by the reduced tse logic power consumption which will be obtained when the electroluminescent array is deactivated. The power consumption of a deactivated tse device could be reduced to essentially zero by allowing the control signal to power-down the logic circuit portion of the tse device as well as the electroluminescent output array.

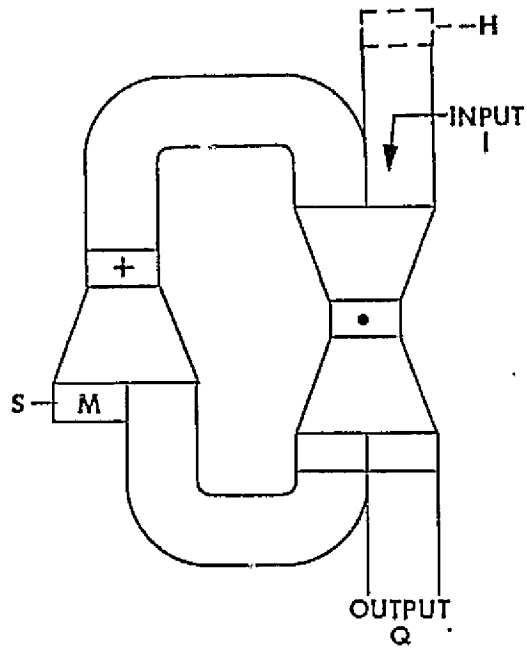
An improved one tse AND latch which uses the proposed control technique is illustrated in Fig. 5.3. Note that control signal S is interfaced to the feedback tse data path using the same technique employed in the basic one tse AND latch (Figure 4.2, page 53). This control signal interface could be simplified by assuming that some active tse logic devices are available which are enabled for normal operation or forced to produce an all logic one output tse depending upon the state of a control line. An increase in the complexity of the active tse logic devices would be required. Since this increase will not be associated with additional advantages such as reduced power consumption, the assumption that this class of tse logic devices will be available will not be made at this time. After additional experience has been gained in the design of tse logic



CONTROL TABLE

E	C	Q^{+}_{ij}
0	0	0
0	1	Q_{ij}
1	0	I_{ij}
1	1	$Q_{ij}+I_{ij}$

Figure 5.2 Improved one tse OR latch.



CONTROL TABLE

H	S	Q_{ij}^{+}
0	0	$Q_{ij} \cdot I_{ij}$
0	1	I_{ij}
1	0	Q_{ij}
1	1	1

Figure 5.3 Improved one tse AND latch.

circuits and in the fabrication of tse devices, a decision can be made concerning the development of tse devices with this capability.

Some Additional Improved Tse Memories

The circuit schematic and timing diagram for an improved one tse master-slave memory is presented in Figure 5.4. Only ten unit gate delays are required to store a new image in this memory whereas 15 unit delays are required to store an image in the basic master-slave tse memory (Figure 4.3, page 55). The cost function of the improved memory is 6A + 4B compared to a cost function of 14A + 8B for the basic master-slave memory.

Master-slave tse memories can be chained to create tse shift registers of any desired length. An example is the six tse circular right shift register illustrated in Figure 5.5. Nine active and six passive tse devices are required in each section of this register for a total cost function of 54A + 36B. Figure 5.6 illustrates the typical control sequences for this shift register. A functionally equivalent shift register constructed to use the elementary control technique would require 21 active and 12 passive tse devices in each section for a total hardware cost of 126A + 72B. The hardware cost of this circuit is reduced by more than 50 percent by adding control lines to the active tse devices.

An alternate implementation of the six tse, circular right shift, parallel-input, parallel-output shift register is shown in Figure 5.7. This design does not exhibit master-slave capability on the parallel inputs; however, as illustrated in Figure 5.8, this design does have the advantage of parallel loading in four unit gate delays rather than the

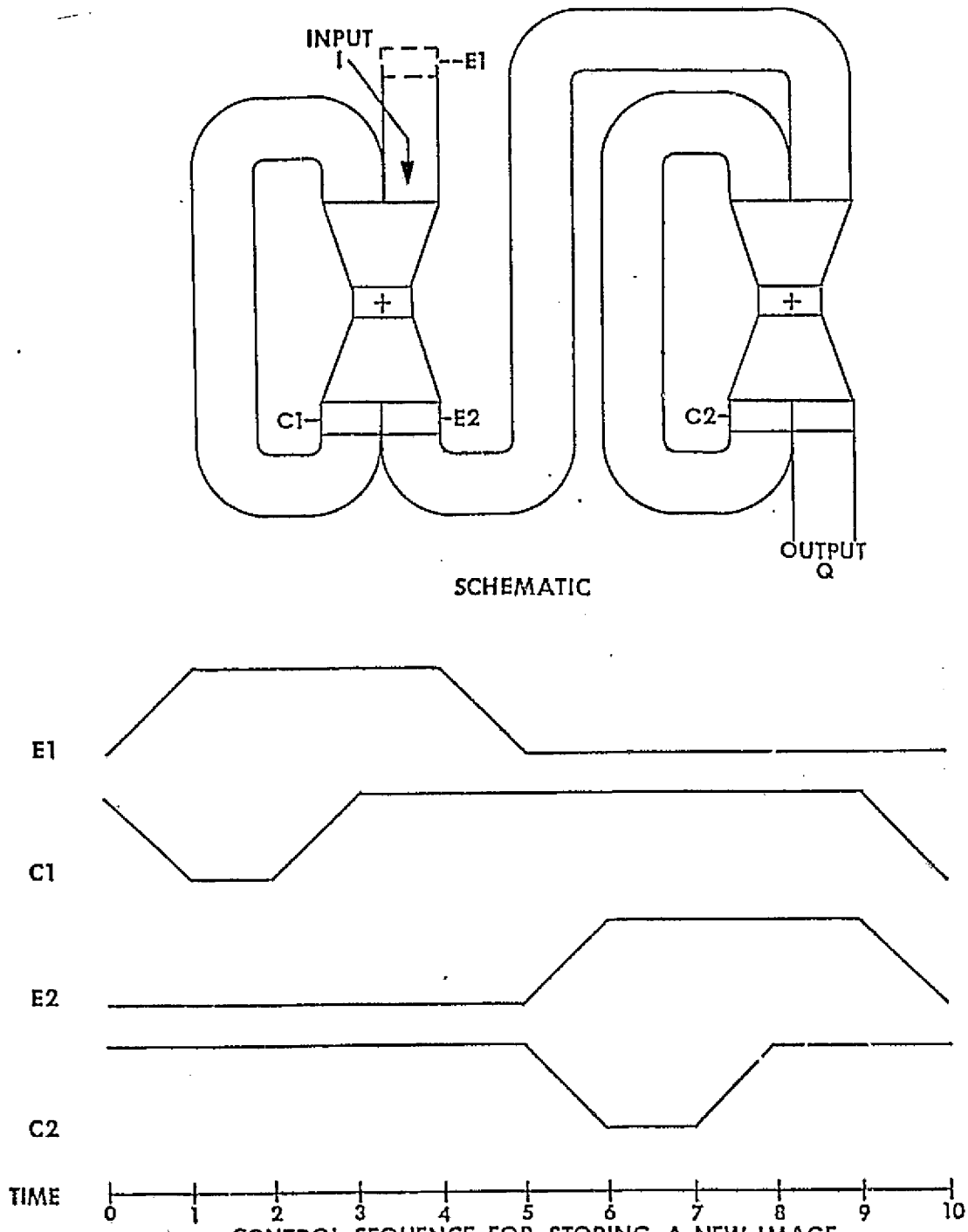


Figure 5.4 Improved master-slave tse memory.

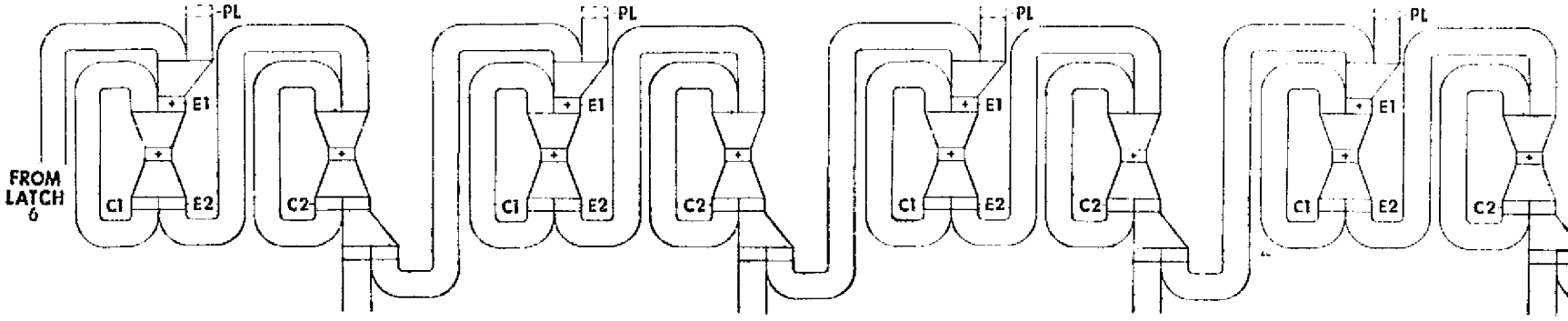
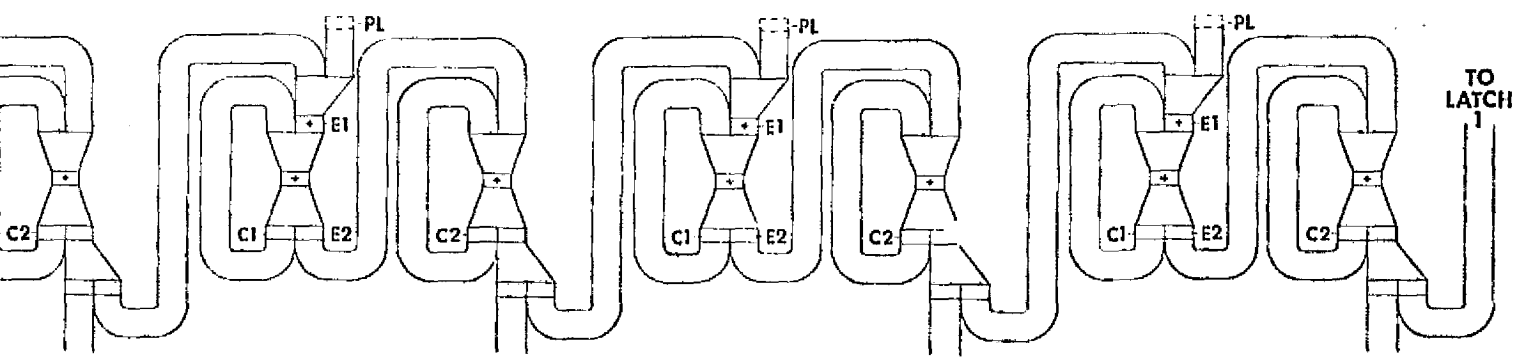


Figure 5.5 Six tse circular right shift parallel-input, parllel-output

FOLDOUT FRAME



hift parallel-input, parllel-output master-slave shift register.

TO DOUT FRAME 2

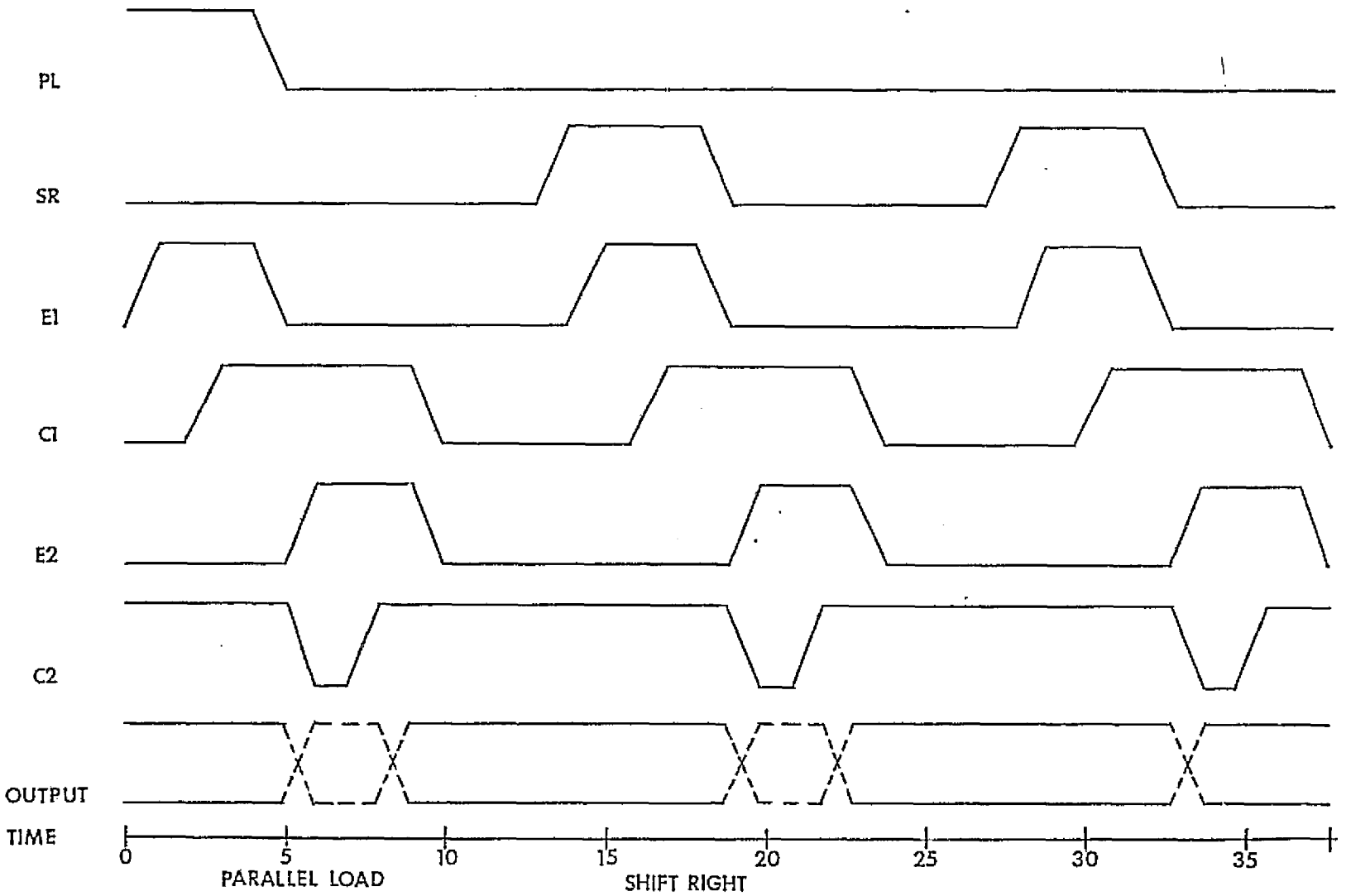


Figure 5.6 Control signal timing diagram for the parallel-input, parallel-output master-slave tse shift register.

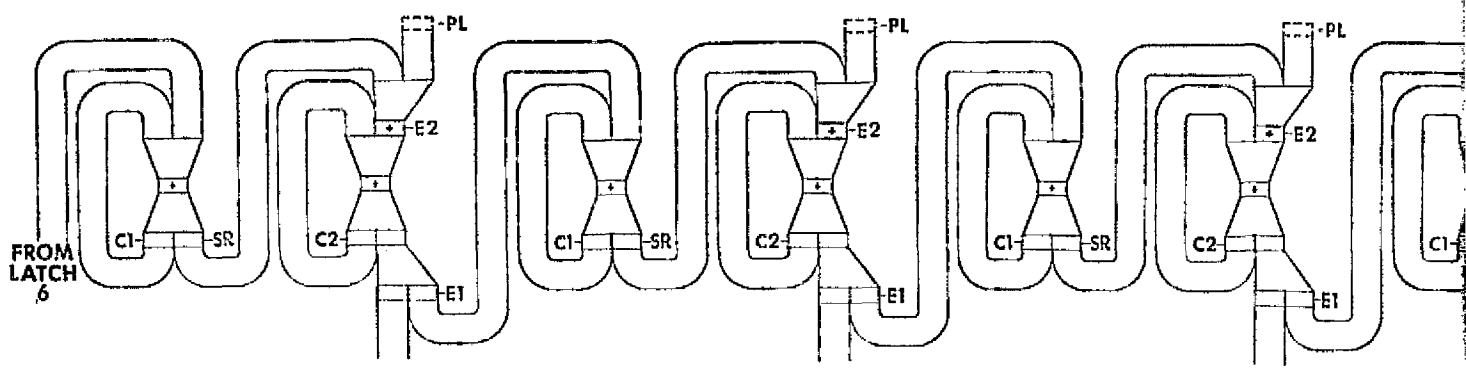
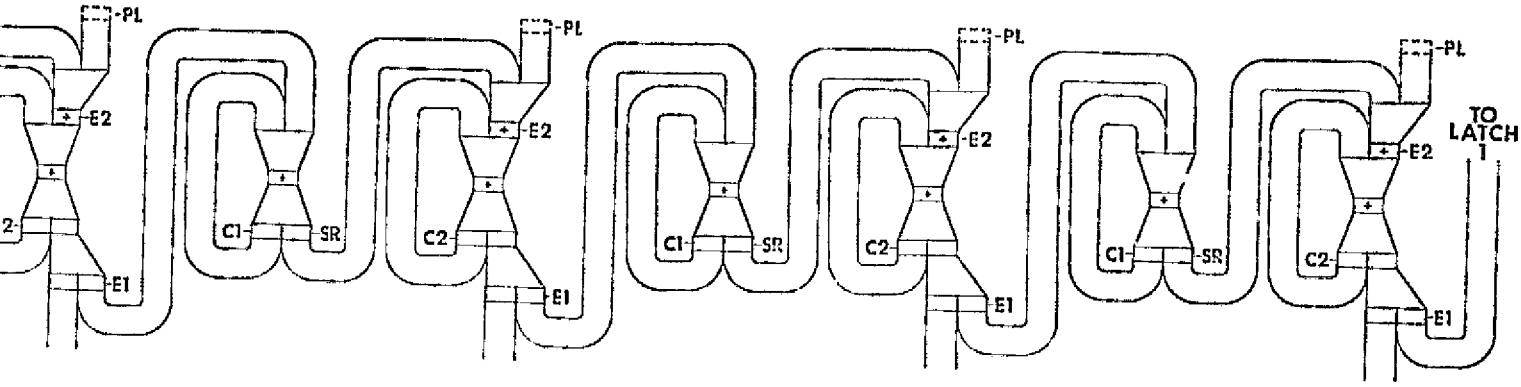


Figure 5.7 Six tse circular right shift, parallel-input,



parallel-input, parallel-output shift register.

FOLDOUT FRAME 2

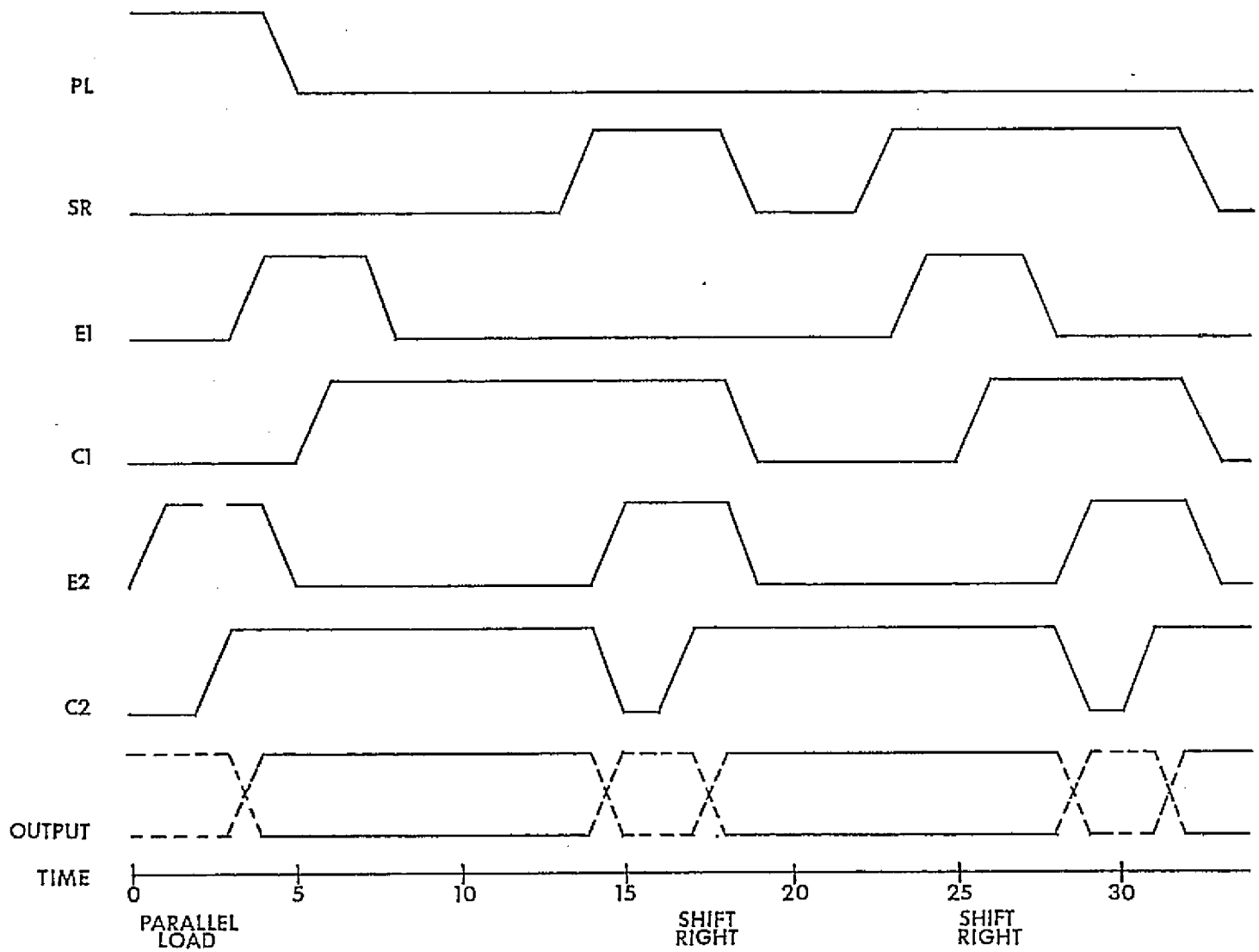


Figure 5.8 Control signal timing diagram for the parallel-input, parallel-output tse shift register.

nine unit gate delays required by the true master-slave design. The same number of tse components are required to construct either circuit.

Improved Index Recognition Circuits

As demonstrated in Chapter 4, index recognition circuit characteristics are important factors in determining the hardware cost and data rate of a tse logic implementation of the skeletonizing algorithm. The space iterative index recognition circuit (Figure 4.11, page 70) requires only ten unit gate delays to recognize basis points with a surround of index one, two, or three but has a hardware cost function of 125A + 77B. A number of index recognition circuits which offer a range of choices in the trade-off between circuit complexity and operating speed are desirable since component minimization is an important goal in the development of tse logic circuits. The basic technique for reducing the amount of hardware required to implement an index recognition circuit is to design the circuit to operate in a time sequential mode.

Figure 5.9 shows an index recognition circuit which identifies basis points with a surround of index one, two, or three by checking for one possible orientation of each of the three indices at a time. The order of the Golay neighbor plane inputs to the combinational logic portion of the circuit is rotated by the multiplexer circuit so that basis points with the correct index will be identified regardless of the orientation of their surround. The partial result obtained after each rotation is stored in the OR latch. As demonstrated by the timing diagram in Figure 5.10, 75 unit delays are required to recognize all three indices using this circuit. The cost function for the multiplexed index recognition circuit is 104A + 69B. Thus, this circuit requires 21 fewer active

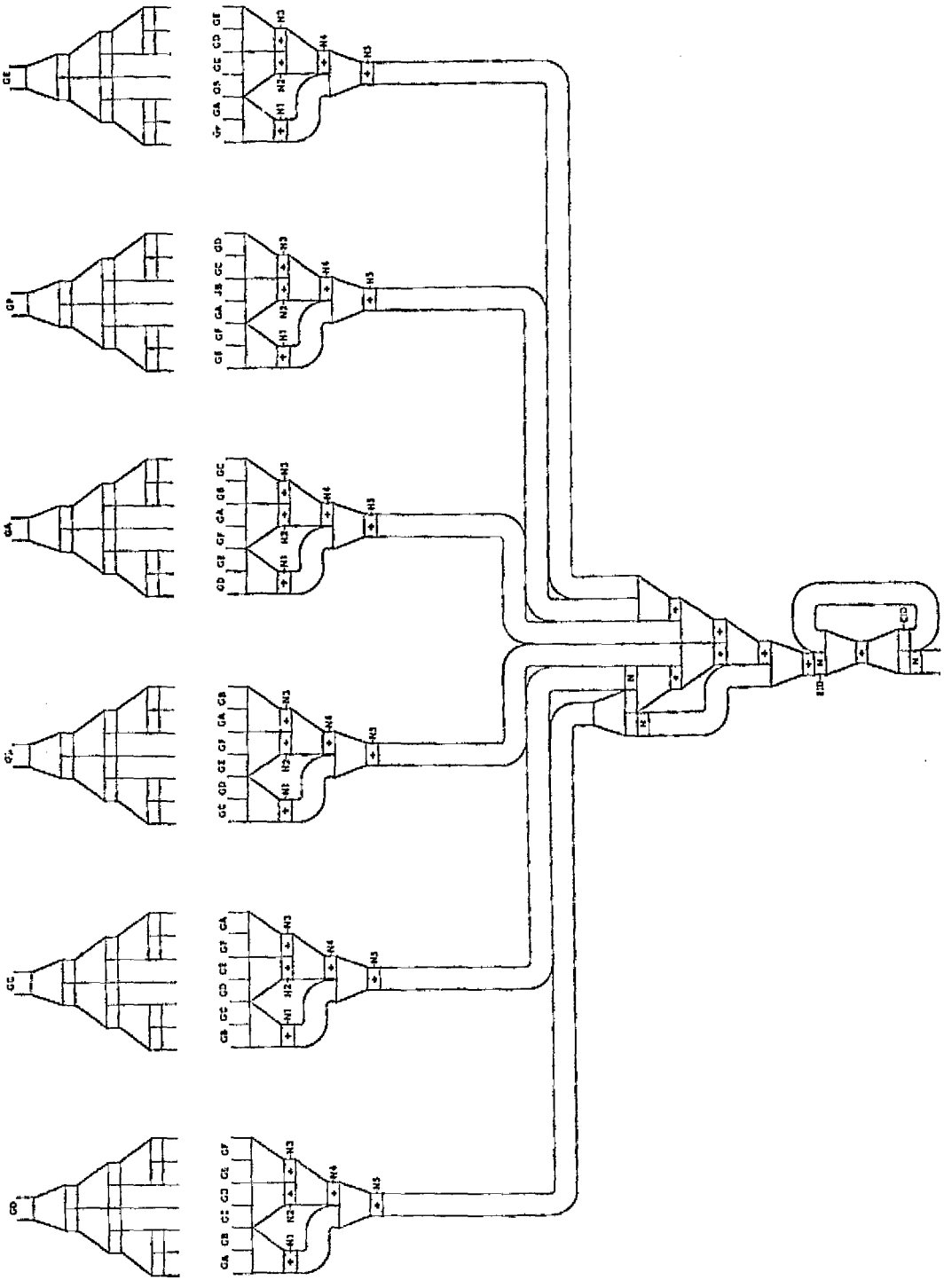


Figure 5.9 Multiplexed index recognition circuit.

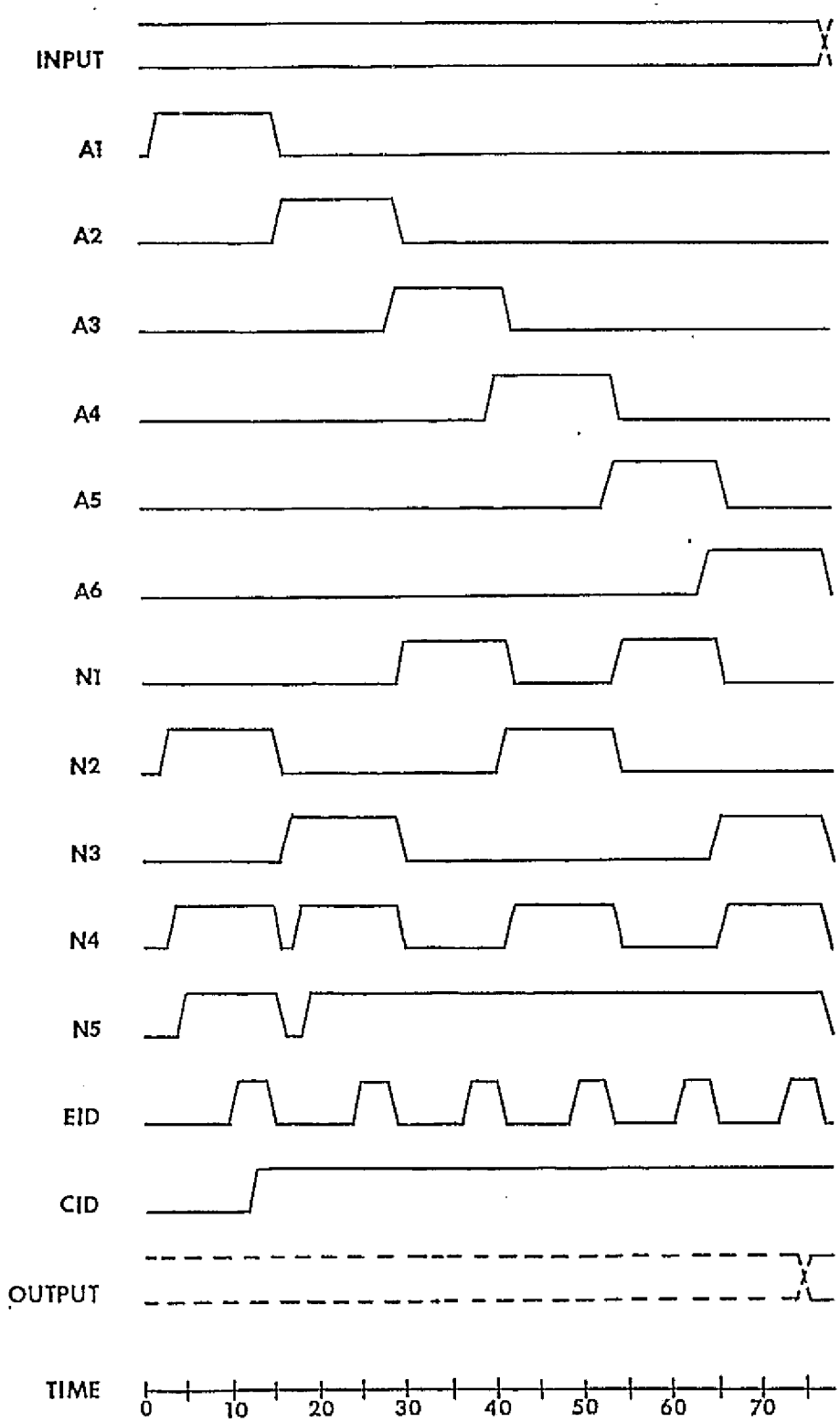


Figure 5.10 Timing diagram for the multiplexed index recognition circuit.

devices and eight fewer passive devices than the space iterative index recognition circuit. The multiplexed index recognition circuit is not practical for implementation using basic tse devices which lack a single line disable input since a minimum of 38 control line interfaces would be required. This would increase the circuit cost function to 180A + 107B which is 42 percent more components than the high performance space iterative index recognition circuit (Figure 4.11, page 70) requires.

An alternate technique for rotating the Golay neighbor planes is illustrated by the shift register based index recognition circuit presented in Figure 5.11. This circuit can recognize all basis points with a surround of index one, two, or three in 82 unit gate delays (Figure 5.12). The cost function for the shift register based index recognition circuit is 68A + 45B. Thus, a 34 percent savings in tse components can be realized with only a nine percent increase in propagation delay by rotating the Golay neighbor planes with a shift register rather than a multiplexer. The shift register type index recognition circuit can be realized using the basic control technique (Figure 5.1, page 87) at a hardware cost of 140A + 81B. Such a design would be impractical, however, since improved performance could be acheived at a lower hardware cost using the space iterative index recognition circuit (Figure 4.11, page 70).

The hardware cost of performing the index recognition task using the comparison type circuit (Figure 4.12, page 71) discussed in Chapter 4 is 60A + 33B. As illustrated in Figure 5.13, the hardware cost of the comparison type index recognition circuit can be reduced to 20A + 13B if integrated circuit EXCLUSIVE-OR gates are available. Because of the general usefulness of the EXCLUSIVE-OR function, EXCLUSIVE-OR gates are proposed for development as the first complex, integrated tse logic device.

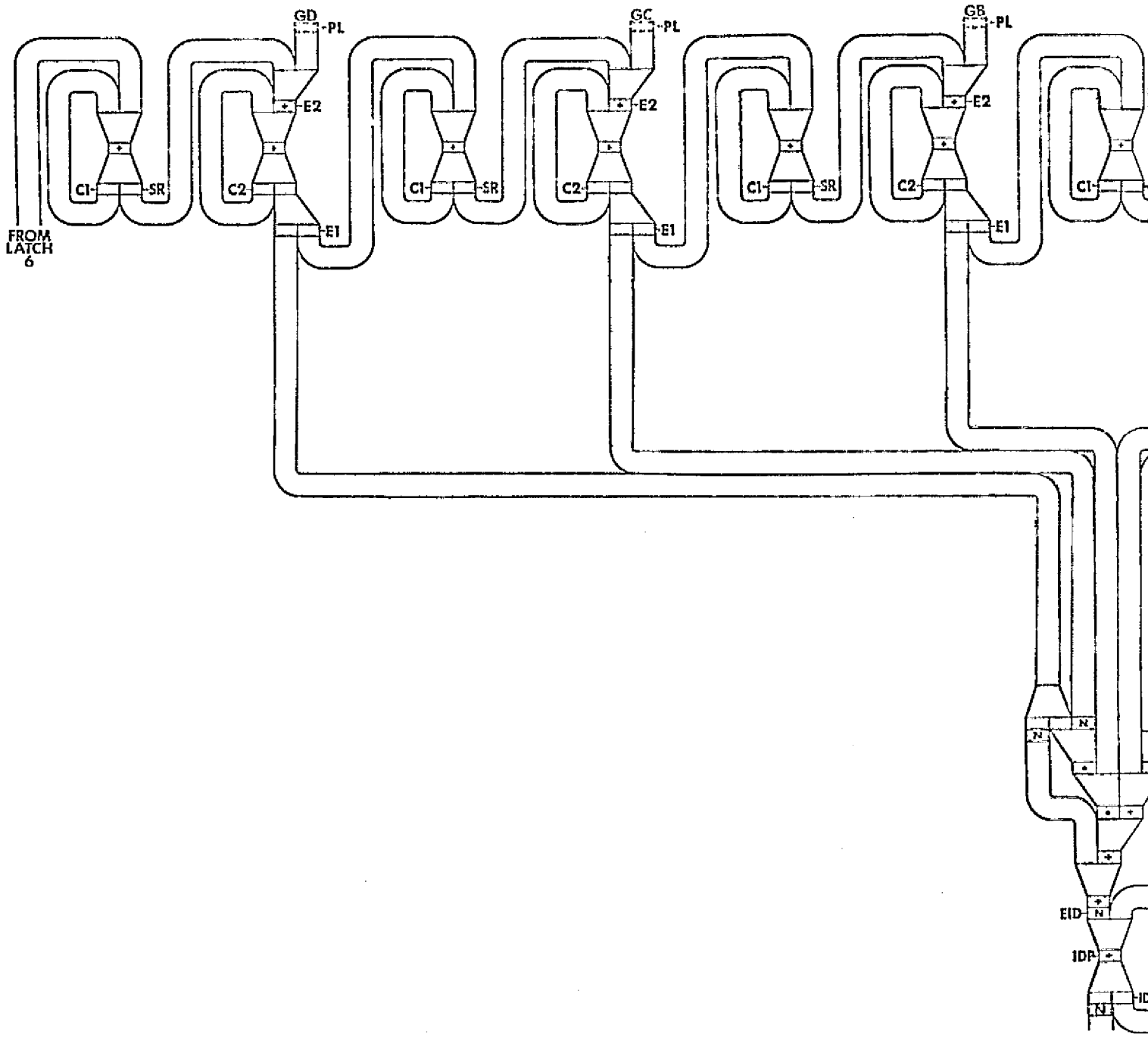
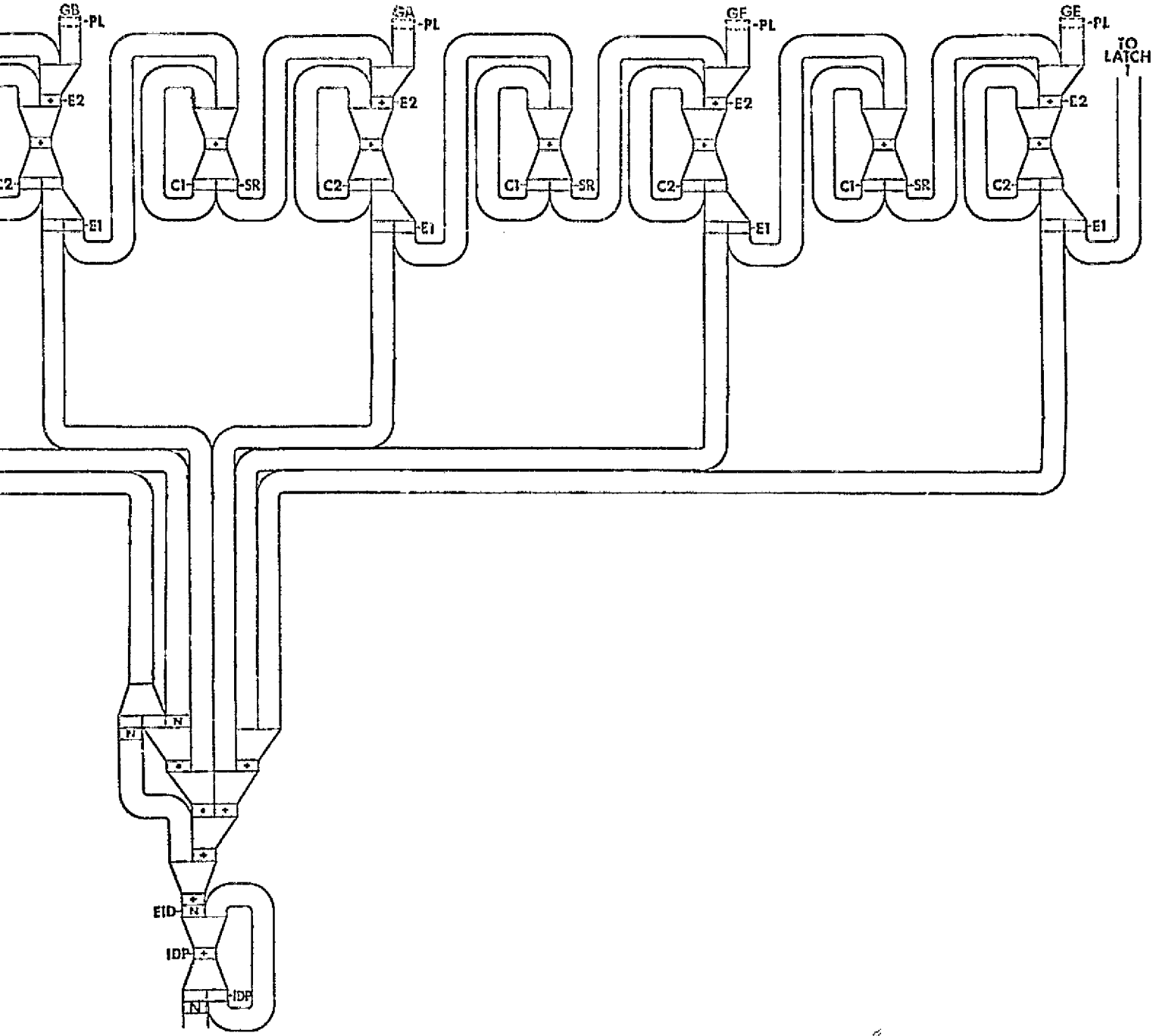


Figure 5.11 Shift register based index r

FOLDOUT FRAME



egister based index recognition circuit.

FOLDOUT FRAME 2

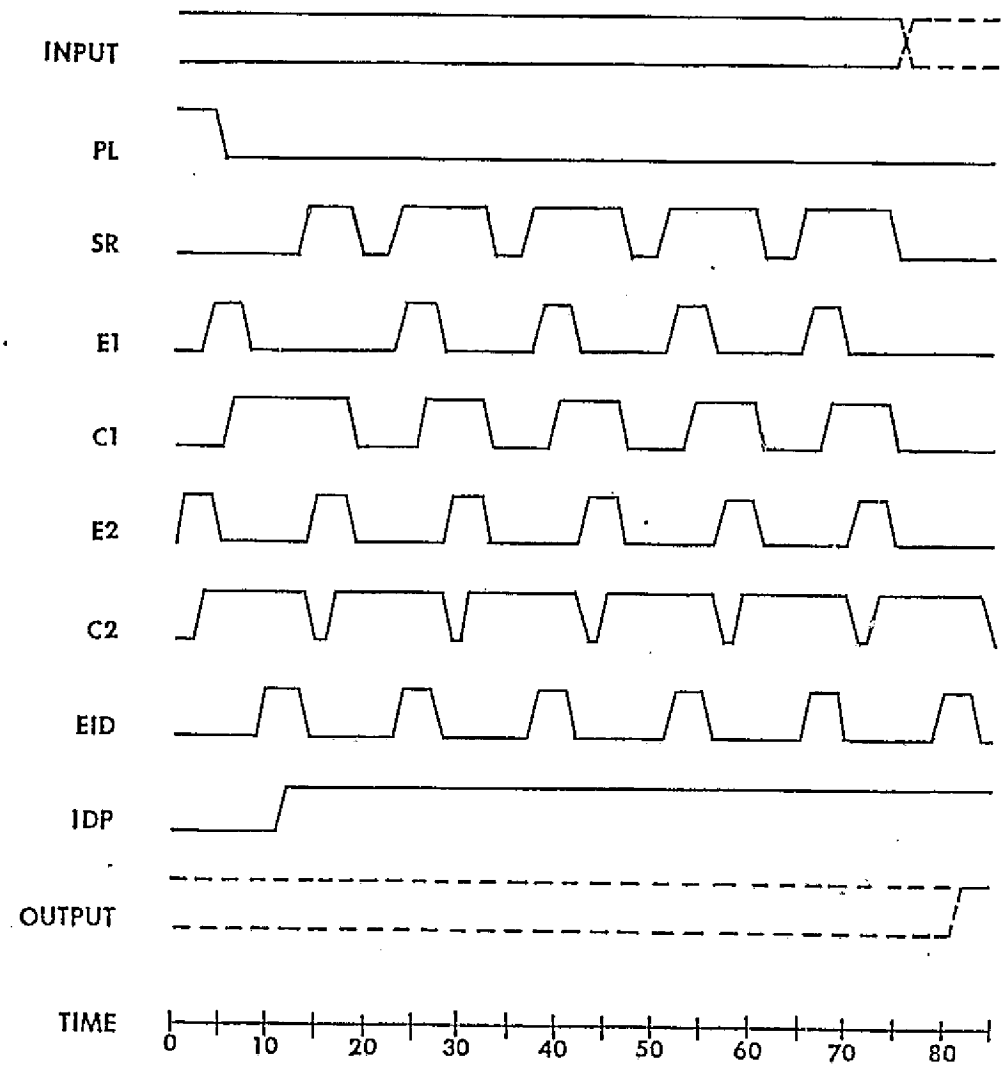


Figure 5.12 Timing diagram for the shift register based index recognition circuit.

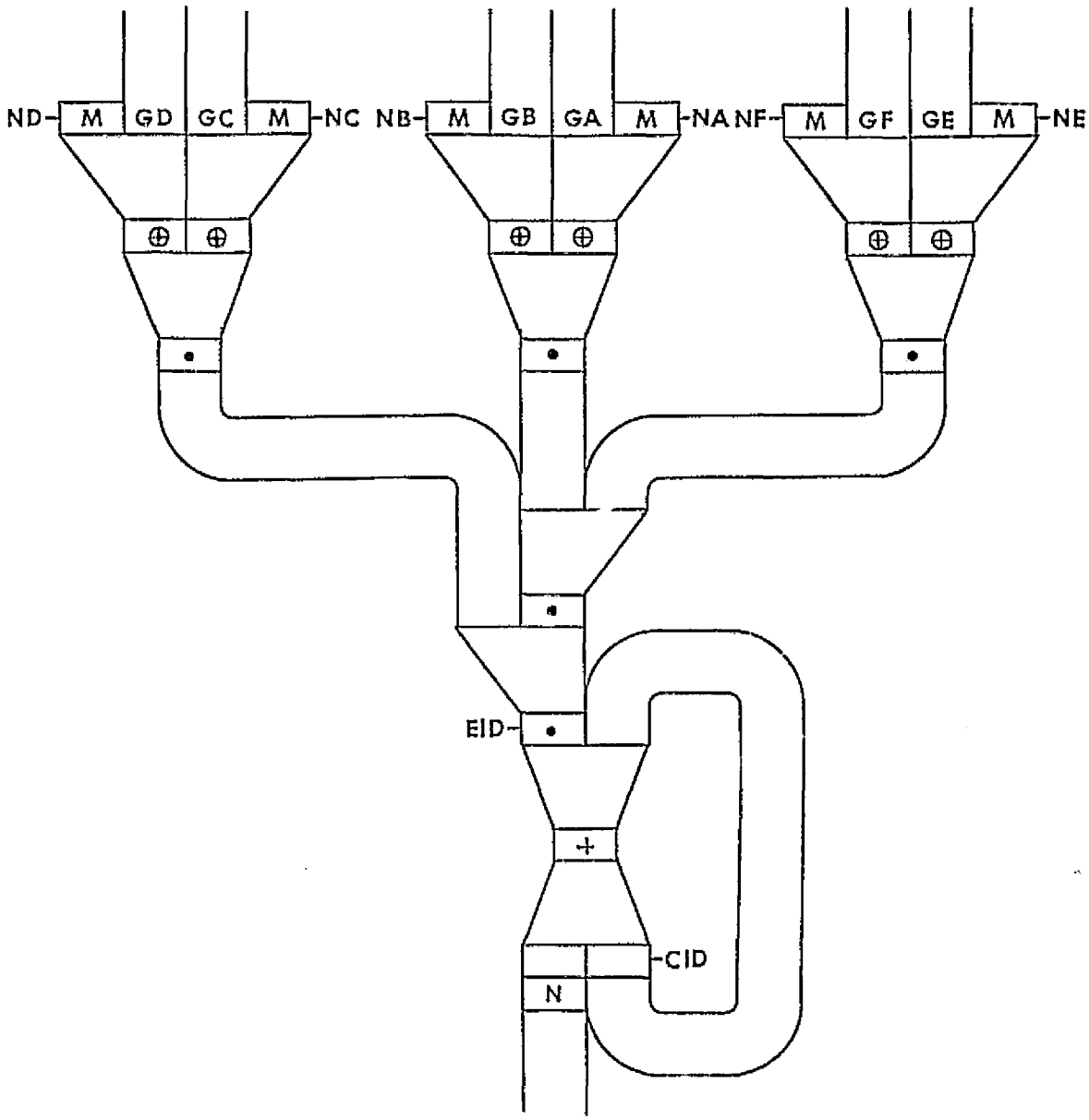


Figure 5.13 Comparison type index recognition circuit using EXCLUSIVE-OR gates.

An improved comparison type index recognition circuit using EXCLUSIVE-OR gates requires only 159 unit gate delays (Figure 5.14) to identify all basis points with a surround of index one, two, or three. The worst case time required to identify basis points with any set of indices is only 285 unit gate delays.

Performance characteristics for the various index recognition circuits that are useful in performing the Golay transform skeletonizing algorithm are summarized in Table 5.1. The trade off between circuit complexity and operating speed is illustrated by the increase in propagation delay as the number of tse components required to perform the index recognition task is reduced. Table 5.1 also shows the potential value of including a single line disable on the active tse logic devices since only two of the five index recognition circuits are practical if the single line disable is not provided.

An OR Latch Implementation of the Skeletonizing Algorithm

A medium performance implementation of the Golay transform skeletonizing algorithm using OR latches and the shift register type index recognition circuit is illustrated in Figure 5.15. This circuit operates in basically the same manner as the skeletonizing machine described in Chapter 4. The OR latch design, however, has the advantage of permitting changes in the subfield operating order. Normally the modular operation specified by the Golay transform skeletonizing algorithm is performed on basis points in subfields one, two, or three in that order. If some other subfield operating order is employed, a skeleton of the original image will still be obtained but, depending upon the shape of the original image, the skeleton might be shifted to a slightly different

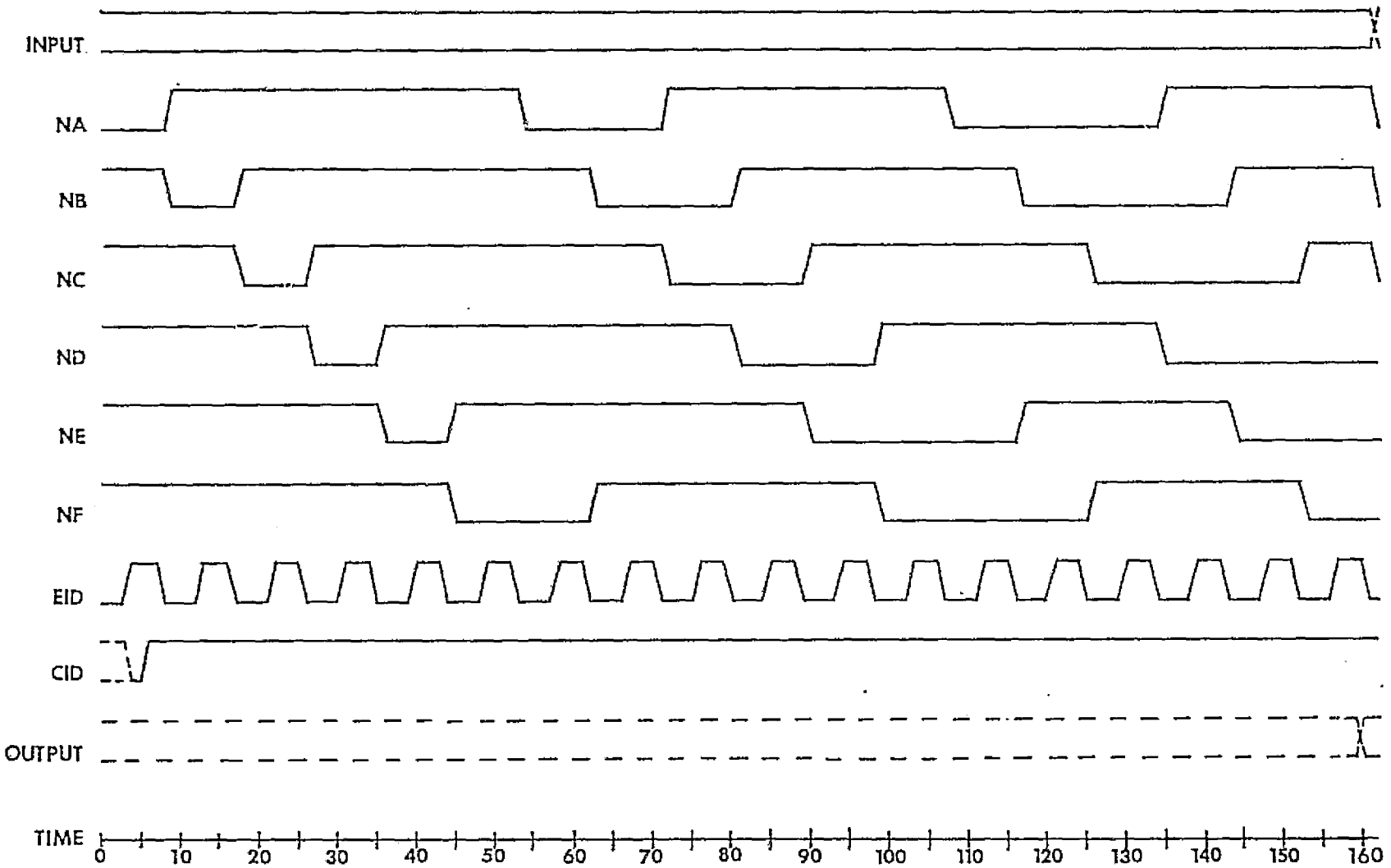


Figure 5.14 Timing diagram for the comparsion type index recognition circuit using EXCLUSIVE-OR gates to identify basis points with an index of one, two, or three.

TABLE 5.1 PERFORMANCE CHARACTERISTICS OF THE INDEX RECOGNITION CIRCUITS

Circuit Type	Figure Number	Cost Function	Propagation Delay in Unit Gate Delay
SPACE ITERATIVE COMBINATIONAL CIRCUIT	4.11	125A+77B	10
MULTIPLEXED CIRCUIT	5.7	104A+69B	75
SHIFT REGISTER CIRCUIT	5.11	68A+45B	82
BASIC COMPARISON CIRCUIT	4.12	60A+33B	214
COMPARISON CIRCUIT WITH EXCLUSIVE-OR GATES	5.13	20A+13B	159

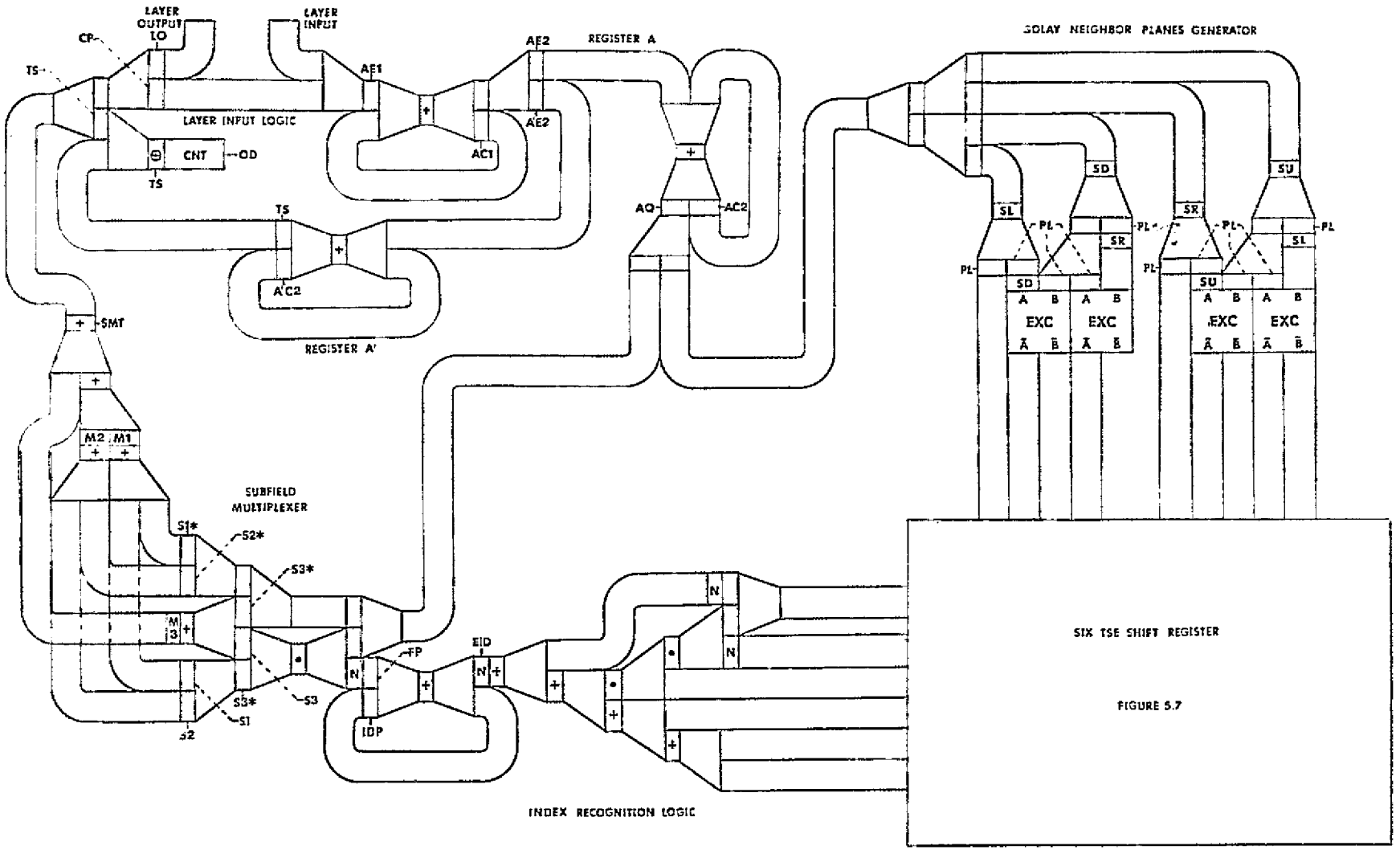


FIGURE 5.7

Figure 5.15 Hardwired skeletonizing machine using OR latches.

location within the image field. This effect could be useful in a potential cloud tracking application which has been suggested for the skeletonizing machine [17].

The location and velocity of major cloud formations is one type of valuable meteorological information which can be obtained from earth resources satellites. Traditionally, successive images of the cloud formation are transmitted to ground stations where conventional computers are used to compute the speed and direction of the cloud formation. If a tse logic skeletonizing machine is included in the satellite, cloud images could potentially be skeletonized in real time. A sequence of skeletons could be transmitted to earth as one image which would show the track of the air mass carrying the cloud. Since less data would be transmitted to earth, the bandwidth of the transmission channel and the processing load on earthbound computers could both be reduced. One potential problem is the time varying shape of the cloud formation. The skeletonizing operation should consistently produce skeletons at a point near the geometric center of the original cloud image to facilitate accurate cloud tracking. An OR latch skeletonizing machine design such as the one illustrated in Figure 5.15 would allow several skeletons of each cloud image to be obtained using different subfield operating orders. The average skeleton of the image would then provide a more consistent indication of cloud position than a single skeleton.

The OR latch skeletonizing machine illustrated in Figure 5.15 is more amenable to alternating subfield operating orders than an improved version of the skeletonizing machine presented in Chapter 4 (Figure 4.17, page 78) would be because the new subfield produced by the current processing step is always combined with the other two subfields via the

subfield multiplexing circuit. There are three alternate paths through the subfield multiplexing circuit. Each path contains a film mask or an active tse OR gate with a programmed electroluminescent output array. The first image path is masked so that only those data points within subfield one of the input tse will be transmitted. Data points within the second and third subfields are zeroed. Similarly, the second and third image paths transmit only data points that are within the second and third subfields of the image, respectively. Either the output of latch A or the Golay function circuit output can be selected for transmission through any of the image paths. Normally, the image output from latch A propagates through the two image paths which correspond to subfields that are not currently being operated on, and the Golay function circuit output propagates through the remaining image path. The image paths are recombined at the output of the subfield multiplexing circuit to produce a new image which is the result of the current skeletonizing algorithm operation. If the current operation is the last of the three subfield operations, the new image can be compared to the result of the preceding operation to determine whether the skeleton is complete or another iteration is required. Image propagation through the three subfield multiplexing circuit image paths is completely controlled by conventional logic signals. Thus, the control unit of the OR latch skeletonizing machine can be designed to select any required subfield operating order and change the order as often as necessary.

The input logic required by the OR latch skeletonizing machine is significantly less complex than the input logic required by the machine presented in Chapter 4. This is partially due to the use of an integrated EXCLUSIVE-OR gate. In addition, however, note that the total

spiller has been replaced by a contractor device which produces a single CMOS compatible output. The output of the contractor is a logic one if any basis point changed states as a result of a modular operation performed during the current iteration of the skeletonizing algorithm. In that case, another iteration is required. The conversion from a total spiller to a contractor is a direct consequence of the assumption that active tse components can be deactivated by a one bit control line. As a further result of this assumption, some of the input logic operations can be performed by the conventional logic control unit, and fewer tse components are required to implement the input logic.

Control of the OR Latch Skeletonizing Machine

A timing diagram for performing one complete iteration of the skeletonizing algorithm using the OR latch skeletonizing machine (Figure 5.15, page 107) is illustrated in Figure 5.16. The timing diagram includes control signals which are used to minimize power consumption by disabling active tse devices when they are not in use. Each iteration of the skeletonizing algorithm is defined as one machine cycle, and each subfield operation is defined as a subcycle. Thus, there are three subcycles within the 315 unit gate delay machine cycle of the OR latch type skeletonizing machine.

The control unit for the skeletonizing machine can be developed using any of the conventional logic, sequential circuit design techniques. One state-of-the-art control unit design is illustrated in Figure 5.17. Control signals for the skeletonizing machine are generated by programmable logic arrays (PLAs). A binary counter chain which

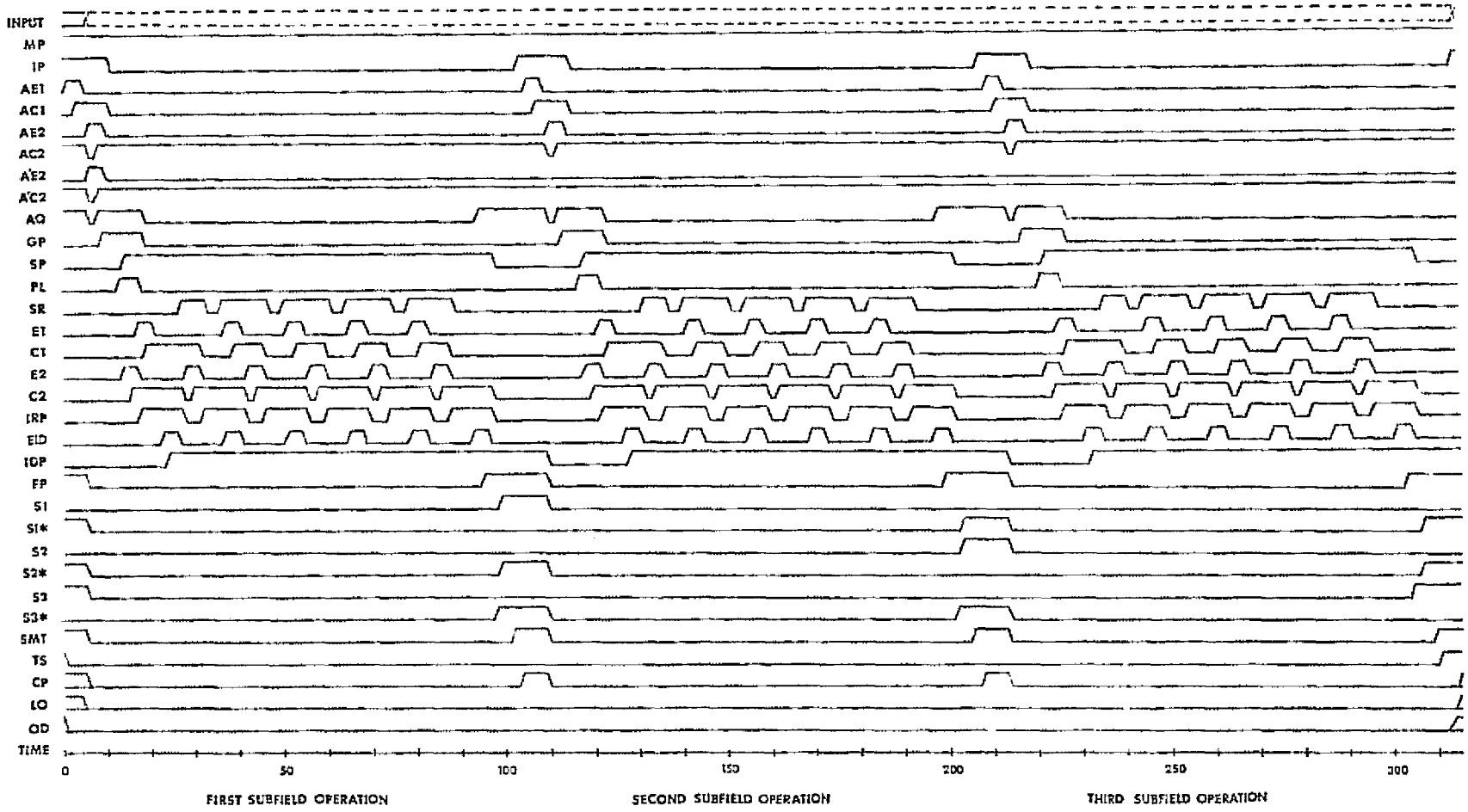


Figure 5.16 Timing diagram for one complete iteration of the skeletonizing algorithm using the OR latch type skeletonizing machine.

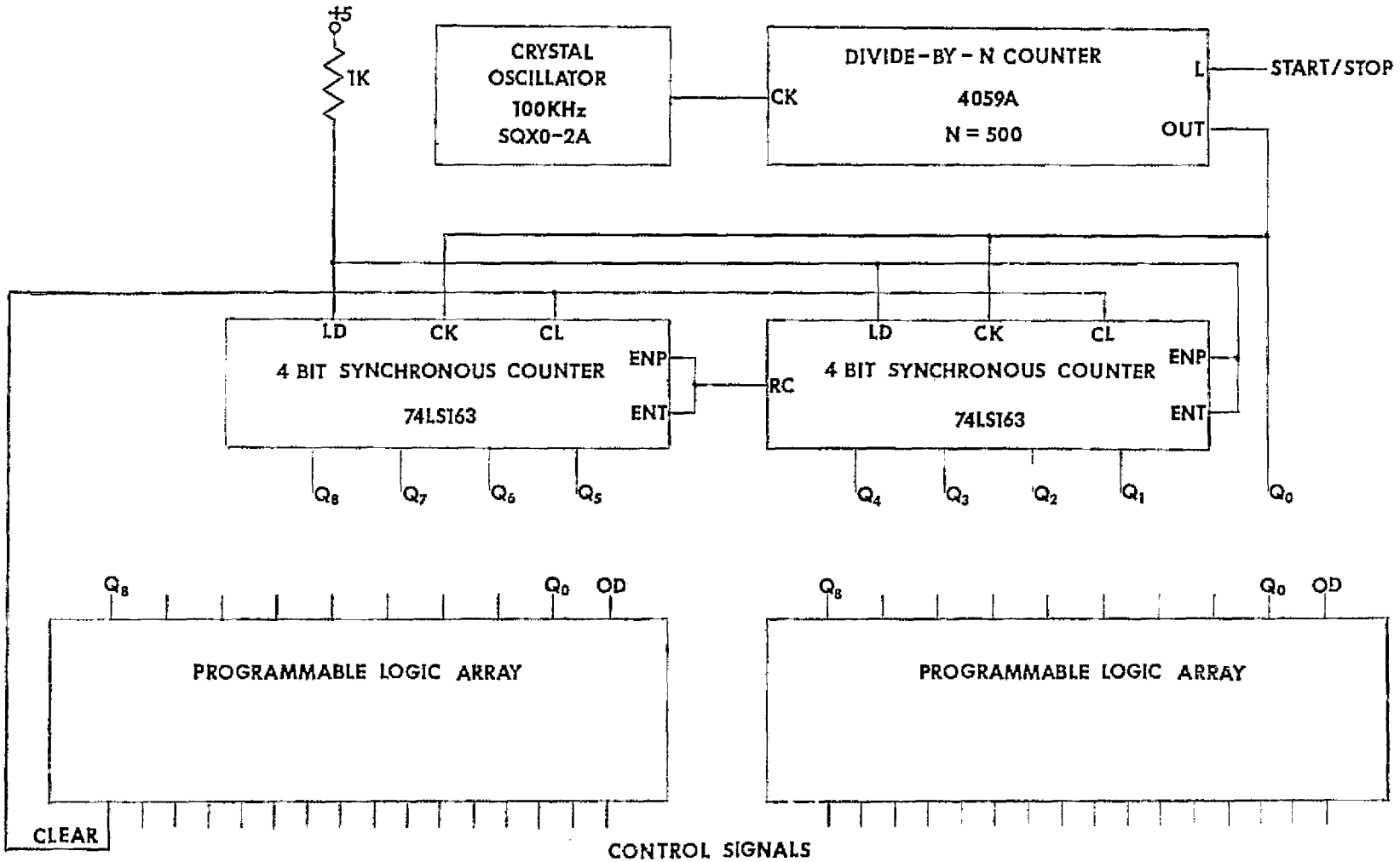


Figure 5.17 Control unit for the OR latch skeletonizing machine.

is driven at a clock rate corresponding to twice the inverse of the standard gate propagation delay provides inputs to the programmable logic arrays. Each unit time increment in the 3.5 unit gate delay machine cycle is uniquely defined by the state of the counter chain. The PLAs produce the control signals required by the skeletonizing machine by decoding the state of the counter chain, the state of the contractor output, and the state of external control signals which start and stop the machine. An output signal from one PLA resets the binary counter chain at the end of each machine cycle. In applications such as cloud tracking where a fixed sequence of different subfield operating orders is required, an additional binary counter circuit can be used to determine the current subfield operating order. The programmable logic array would provide a clock pulse to the counter at the completion of each machine cycle. The PLA would then decode the new counter state to determine the subfield operating order required during the next machine cycle. Table 5.2 summarizes the function of each control signal produced by the OR latch skeletonizing machine control unit.

An Improved AND Latch Implementation of the Skeletonizing Algorithm

The OR latch type skeletonizing machine can be converted to an improved version of the AND latch type skeletonizing machine presented in Chapter 4 by redesigning register A, register A', and the subfield multiplexing circuit. A schematic for the AND latch type skeletonizing machine is presented in Figure 5.18, and a timing diagram for one machine cycle is illustrated in Figure 5.19. Since a higher data rate is achieved with fewer tse components, this design is preferred over the

TABLE 5.2

OR LATCH SKELETONIZING MACHINE CONTROL SIGNALS

Control Signal	Number of Components Controlled	Control Function
MP	2	Machine Power
IP	3	Input Logic Power
AE1	1	Layer Input Gate
AC1	1	Latch A
AE2	1	Latch A
AC2	1	Latch A
A'E2	1	Latch A'
A'C2	1	Latch A'
AQ	1	Latch A Output
GP	9	GNPG Power
SP	18	Shift Register Power
PL	10	Shift Register Input
SR	6	Shift Slave into Master
E1	6	Shift Right to Next Latch
C1	6	Store Slave
E2	6	Input to Master
C2	6	Store Master
IRP	16	Index Recognition Combinational Logic Power
EID	1	Index Recognition Latch Input
IDP	2	Index Recognition Latch Power
FP	10	Golay Function and Multiplexer Power
S1	1	First Subfield Control
S1*	1	NOT First Subfield Control
S2	1	Second Subfield Control
S2*	1	NOT Second Subfield Control
S3	1	Third Subfield Control
S3*	2	NOT Third Subfield Control
SMT	1	Subfield Multiplexer Output
TS	4	Contractor Power
CP	1	Continue Processing Control
LO	1	Layer Output Control
OD	0	Contractor Output Signal

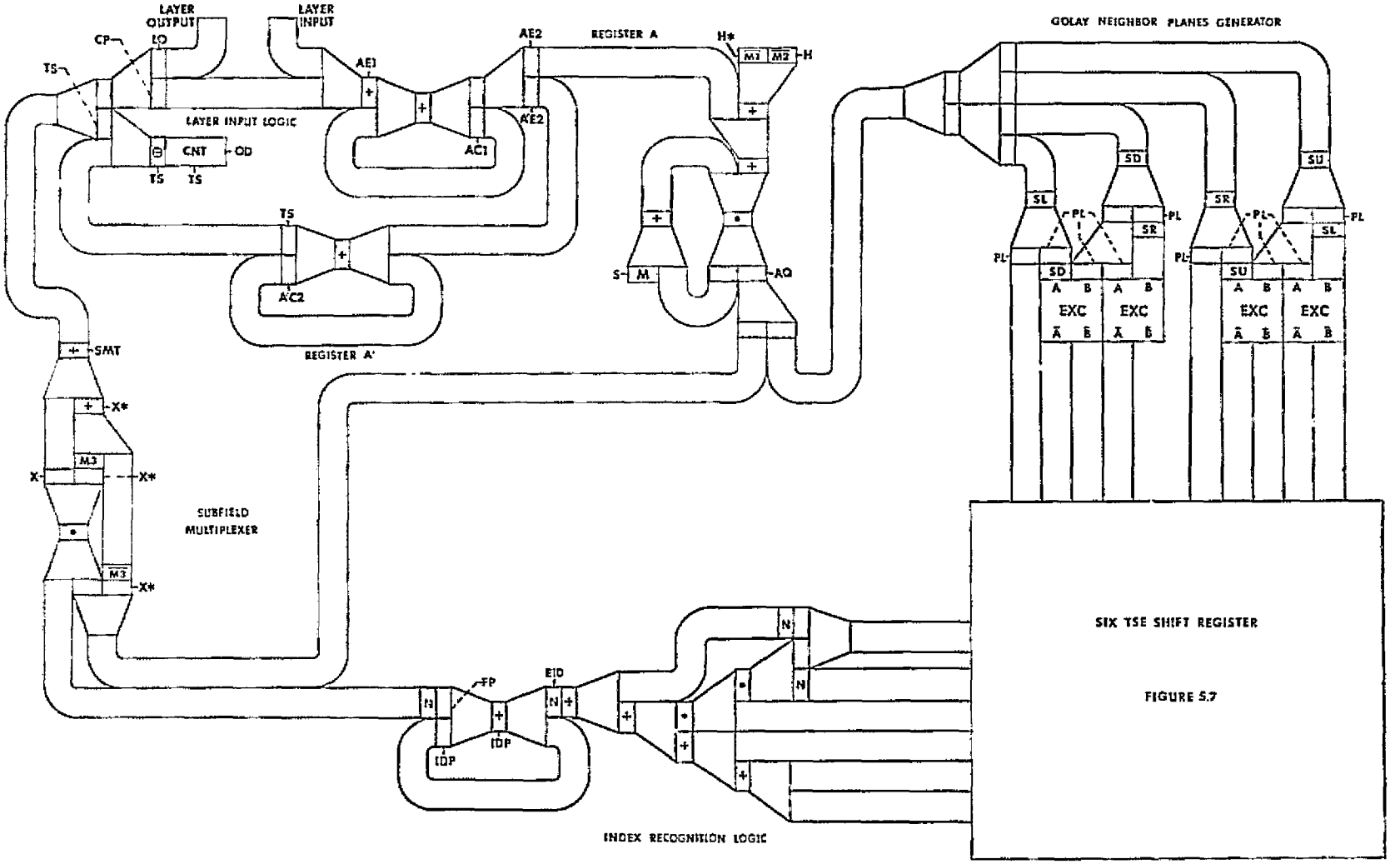


Figure 5.18 Schematic diagram for an improved AND latch hardwired skeletonizing machine.

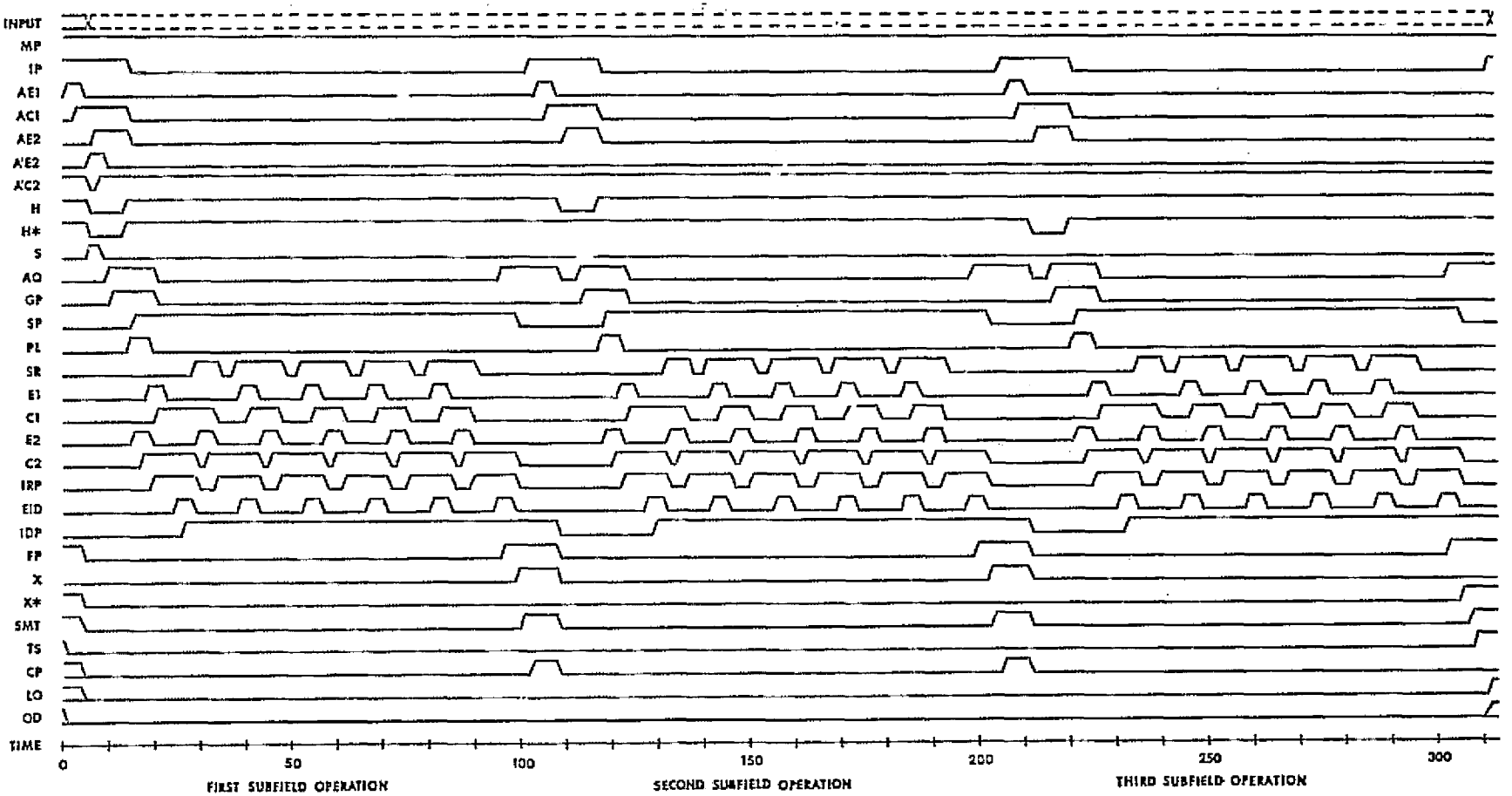


Figure 5.19 Timing diagram for one complete iteration of the skeletonizing algorithm using the improved AND latch skeletonizing machine.

OR latch design when only one subfield operating order is required. A control unit for the AND latch machine could be constructed using the technique described for the OR latch machine. Control signals for the AND latch skeletonizing machine are described in Table 5.3. The hardware costs, power requirements, and performance of the AND latch and OR latch versions of the skeletonizing machine are summarized in Table 5.4.

The AND and OR latch skeletoninzing machines presented in this chapter are medium performance machines. In particular applications, a higher throughput design or a design which requires fewer tse components may be required. Because the basic skeletonizing machine designs are independent of the type of index recognition circuit used, these machines can be tailored to specific applications by selecting the appropriate index recognition circuit from Table 5.1, page 106. In low data rate applications, the improved comparison type index recognition circuit should be used to reduce hardware costs. For applications where the machine cycle time of the skeletonizing machine using a shift register based index recognition circuit is too long, the multiplexed or space iterative index recognition circuits can be employed at a significantly higher hardware cost. Alternately, the skeletonzing machine organization described in Chapter 7 can be utilized. This organization offers several performance advantages in ultrahigh data rate applications.

TABLE 5.3

IMPROVED AND LATCH SKELETONIZING MACHINE CONTROL SIGNALS

Control Signal	Number of Components Controlled	Control Function
MP	6	Machine Power
IP	3	Input Logic Power
AE1	1	Layer Input Gate
AC1	1	Latch A
AE2	1	Latch A
A'E2	1	Latch A'
A'C2	1	Latch A'
H	1	Latch A Input
H*	1	Latch A Input
S	1	Latch A Store
AQ	1	Latch A Output
GP	9	GNPG Power
SP	18	Shift Register Power
PL	10	Shift Register Input
SR	6	Shift Slave into Master
E1	6	Shift Right to Next Latch
C1	6	Store Slave
E2	6	Input to Master
C2	6	Store Master
IRP	16	Index Recognition Combinational Logic Power
EID	1	Index Recognition Latch Input
IDP	2	Index Recognition Latch Power
FP	5	Golay Function and Multiplexer Power
X	1	NOT First Subfield Control
X*	3	First Subfield Control
SMT	1	Subfield Multiplexer Output
TS	4	Contractor Power
CP	1	Continue Processing Control
LO	1	Layer Output Control
OD	0	Contractor Output Signal

TABLE 5.4

PERFORMANCE CHARACTERISTICS OF THE IMPROVED
AND LATCH AND OR LATCH SKELETONIZING MACHINES

Machine Type	Cost Function	Number of Control Signals	Total Component Count	Gate Delays per Iteration (Time in Seconds)	Data Rate Simple Images per Minute	Average Power Consumption in Watts	Peak Power Consumption in Watts	Speed-Power Product in Watt-Seconds
OR LATCH	123A+89B	31	212	315 (1.58)	38.10	145.70	177.00	229.47
IMPROVED AND LATCH	120A+86B	29	206	313 (1.57)	38.34	160.00	195.00	250.35

CHAPTER 6

A PIPELINED ARCHITECTURE FOR THE SKELETONIZING MACHINE

The medium performance tse logic skeletonizing machines presented in Chapter 5 achieve high data processing rates using a minimum number of elementary tse logic devices. However, certain applications may require even higher data processing rates at the expense of additional hardware. The various index recognition circuit designs described in Chapter 5 illustrate that the data processing rate can be improved significantly by increasing the hardware complexity of the index recognition circuit. This chapter presents a pipelined architecture for the skeletonizing machine which allows further improvements in the data processing rate through a more efficient utilization of the index recognition circuit.

Multiple Tse Processing

The Golay transform algorithms permit the transformation of only one subfield of an image at a time. As a result, only one third or less (less for the four or seven subfields case) of the elements of each tse logic device used in the Golay neighbor planes generator, index recognition, and Golay function circuits can be performing a useful operation on a single image at any instant. The operation performed by the Golay neighbor planes generator inherently restricts the number of simultaneous input images to one because all subfields of the input image are involved in the output function. Points within different subfields do not interact in the index recognition and Golay function

circuits. Therefore, these circuits are capable of processing distinct subfields from several different images simultaneously.

Figure 6.1 illustrates a three plane mixer circuit which can be used to create a composite image for parallel processing by the index recognition circuit. The inputs to the three plane mixer are Golay neighbor planes of the same type from three separate tses. To achieve the minimum index recognition time, the Golay neighbor planes must be available simultaneously, and, thus, three Golay neighbor planes generator circuits are required. The shift register based index recognition circuit (Figure 5.11, p. 101) provides input latches which can accept the three subfields of the composite tse from the three plane mixer sequentially. This permits the use of a single Golay neighbor planes generator circuit when a somewhat longer image processing time is acceptable. Six three plane mixer circuits are required to produce the composite Golay neighbor planes GA*, GB*, GC*, GD*, GE*, and GF*. One additional three plane mixer is used to create a composite image, QA*, of the three tses currently being processed. This image is one input to the Golay function logic.

A Minimum Hardware, Modified Pipeline Implementation of the Skeletonizing Algorithm

The possibility of processing distinct subfields from several different images simultaneously suggest the development of a skeletonizing machine architecture which uses a modification of the pipeline principle. Figure 6.2 illustrates the traditional pipeline orgainzation for an image processing machine. The image processing algorithm is

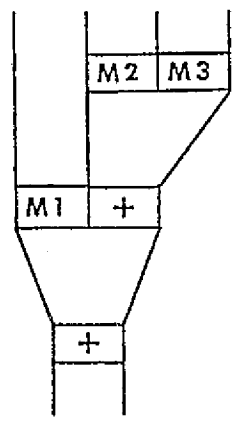


Figure 6.1 Three plane mixer.

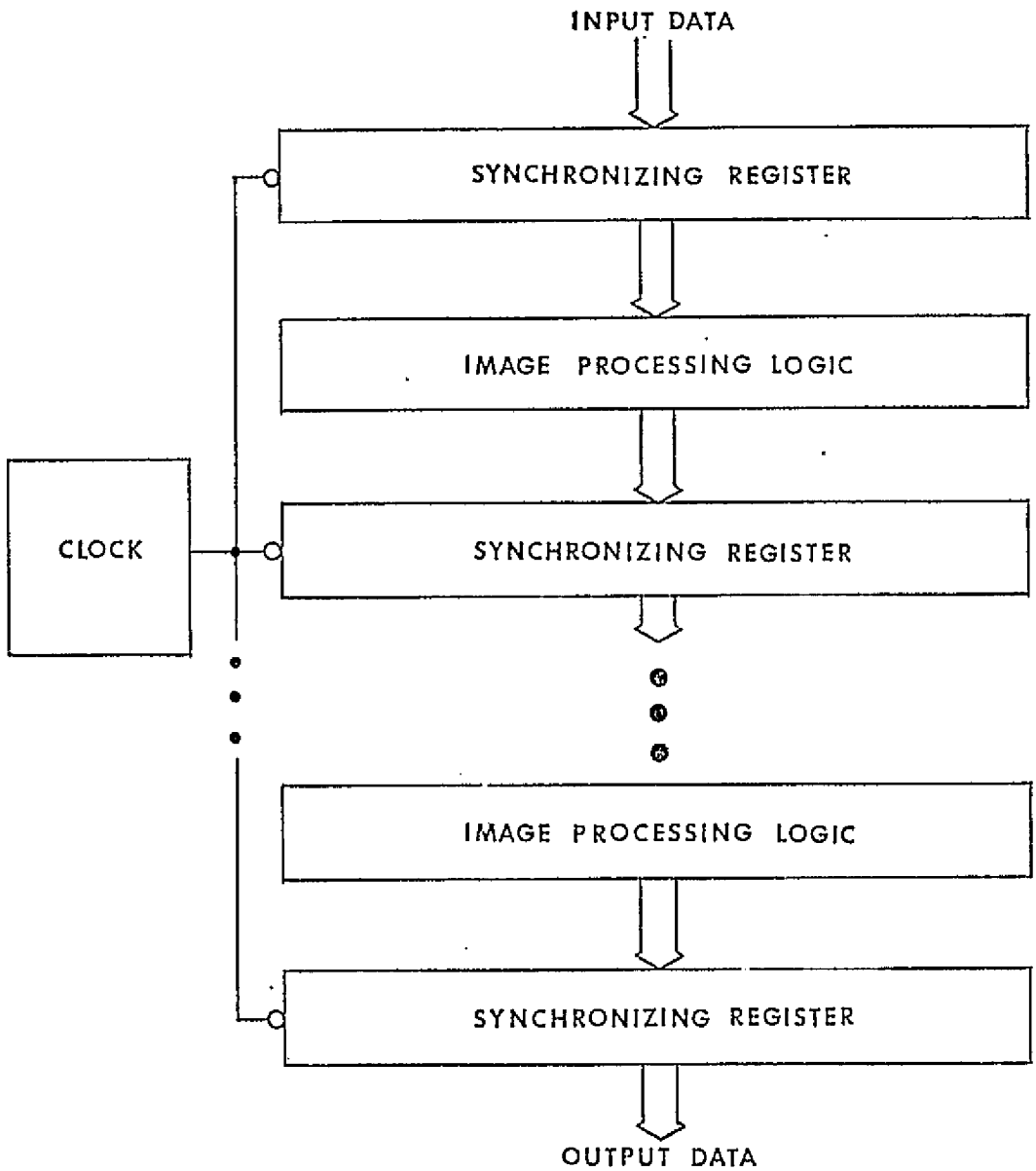


Figure 6.2 Block diagram of a traditional pipeline machine architecture.

broken down into a sequence of steps which can be implemented by successive logic circuits. Synchronizing registers [18] store the partial result obtained from each step while the succeeding processing step is being performed. Images are fed into the top of the pipeline, processed as they are clocked down the pipeline, and extracted at the bottom. This type of machine processes images at the rate at which they can be clocked into the machine rather than at a rate which is dependent upon the complexity of the algorithm. Although this organization is expensive in terms of hardware cost, the pipeline is highly efficient since all of the gating can be utilized 100 percent of the time [18].

Golay transform algorithms are generally not fixed length algorithms because the required number of iterations is a function of the image being processed. An excessively large number of synchronizing registers and logic circuits would be required to guarantee the completion of the Golay transform in a straight pipeline of the type illustrated in Figure 6.2. This difficulty can be overcome by providing a gated feedback path for the output image. A logic circuit at the input of the pipeline would determine whether to gate a new image into the first synchronizing register or gate the current output back into the pipeline for additional processing. Using this technique, any integral number of processing stages can be used to construct a machine to perform a convergent Golay transform such as the skeletonizing algorithm.

Although feasible, a conventional pipeline organization of a Golay transform machine would not be efficient because only one subfield of an input image could be processed at a time. Thus, one-third or more

of the elements of each device employed in the image processing logic would be unused at any instant. Figure 6.3 is a block diagram of a unique pipeline organization for Golay transform processing machines. This organization allows one image processing logic circuit to process one subfield from each of three different pipelined images simultaneously. The hardware minimization achieved by using this architecture is extremely important because of the high cost projected for tse logic devices.

A block diagram of a tse logic implementation of the skeletonizing algorithm using the new pipeline organization is illustrated in Figure 6.4. The Golay neighbor planes generator, three plane mixer, index recognition, and Golay function logic circuits comprise the image processing logic. Three Golay neighbor planes generators are required to obtain the highest processing rates. However, Figure 6.4 illustrates a lower hardware cost design which takes advantage of the latches available in the shift register based index recognition circuit to permit sequential use of a single Golay neighbor planes generator circuit.

Latches A, B, and C are the synchronizing registers of a conventional pipeline organization. One subfield operation is performed on the input image as the image is clocked between each succeeding latch. Two plane mixer circuits, such as the one illustrated in Figure 6.5, are used to form the composite images which represent the result of each subfield operation. In addition to the two plane mixer which provides the feedback signal to the layer input logic, a two plane mixer is required at the input to latch B and latch C. Latches B´ and C´ preserve the images present at the beginning of each iteration of the algorithm for comparison to the processed images.

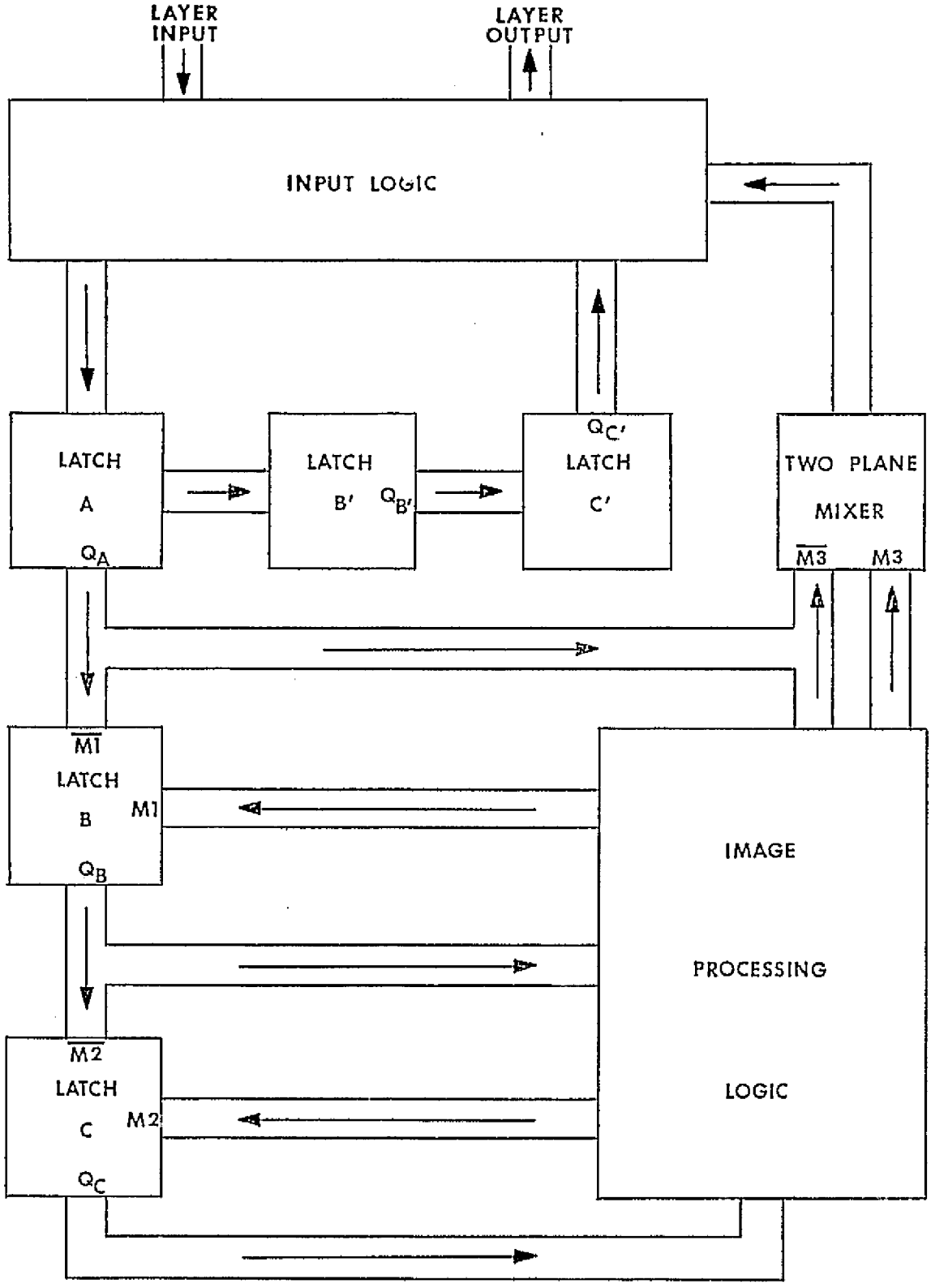


Figure 6.3 Block diagram of a modified pipeline architecture for tse logic image processing machines using Golay transforms.

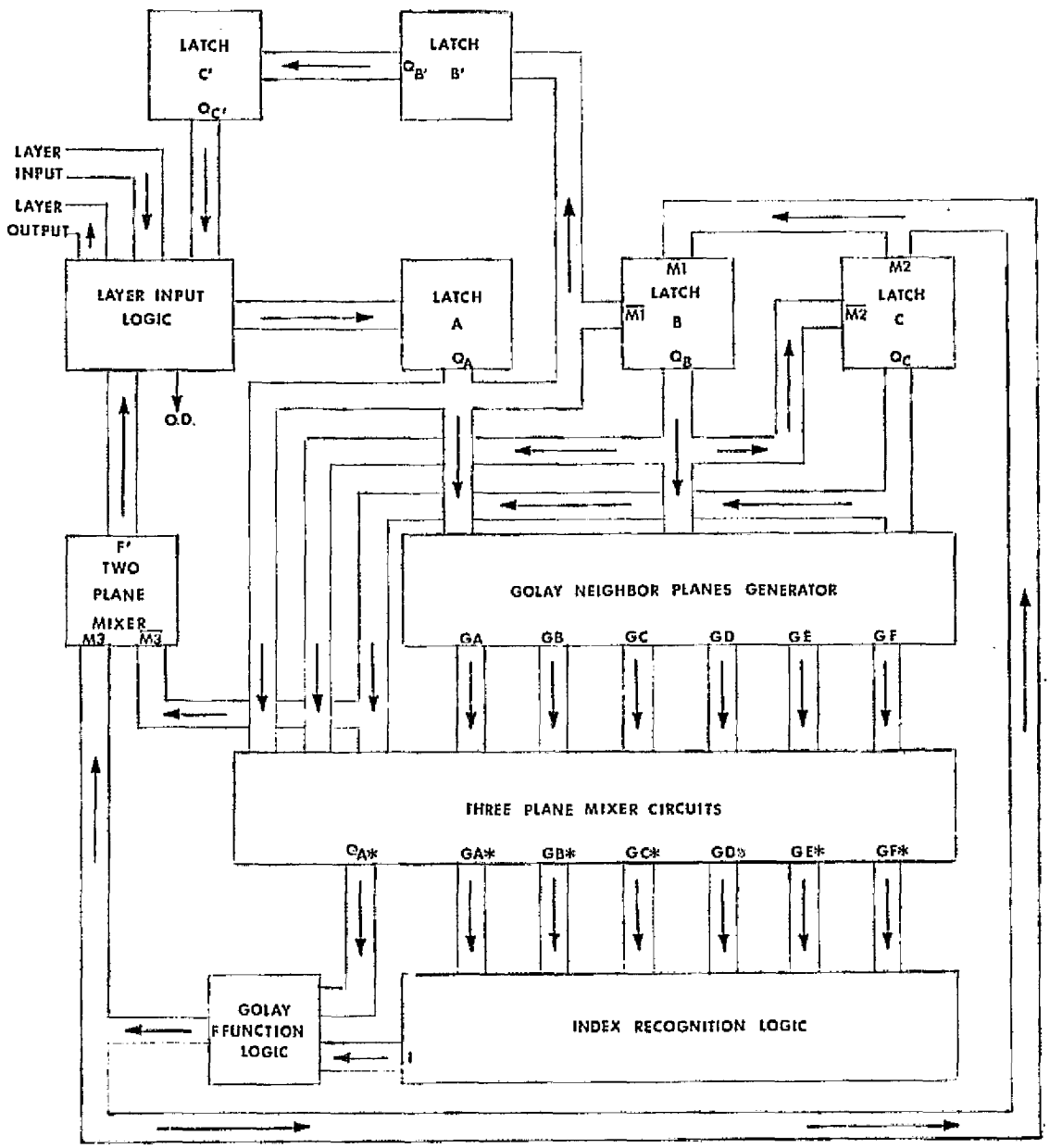


Figure 6.4 Block diagram of a modified pipeline tse logic implementation fo the skeletonizing algorithm.

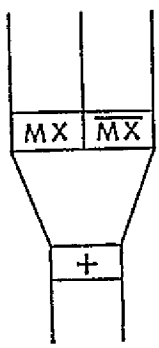


Figure 6.5 A two plane mixer.

The operation of this machine can be explained by following an image, J, through one subfield operation of the skeletonizing algorithm. Image J is gated through the layer input logic and clocked into latch A. Simultaneously, other images are clocked into latches B, B´, C, and C´. Now, assume that latch B´ contains a previous input image, K, latch B contains the result of the first subfield operation on K, K*, latch C´ contains a previous input image, L, and latch C contains the result of the first and second subfield operations on image L, L**. The output of latch A is enabled and Golay neighbor planes are generated for image J. Subfield one of each of the Golay neighbor planes from image J is gated through the three plane mixer network and clocked into the shift register type index recognition circuit. The output of latch A is disabled, and the output of latch B is enabled so that subfield two from each Golay neighbor plane of image K* can be clocked into the index recognition circuit after disabling output QB of latch B and enabling output QC of latch C.

Upon completion of the index recognition procedure, images J, K*, and L** are combined in another three plane mixer, and a composite image consisting of subfield one of image J, subfield two of image K*, and subfield three of image L** is provided to the Golay function logic. Image F, which is formed by the Golay function logic, consists of the first, second, and third subfields of images J, K*, and L**, respectively. The first subfield of image F and the second and third subfields of image J are combined by a two plane mixer for clocking into latch B. Subfield two of image F is combined with subfields one and three of image K* by another two plane mixer for clocking into latch C. Image L** has just completed one iteration of the skeletonizing algori-

thm, so subfield three of image F is combined with the previously processed subfields one and two of image L^{**} to form image F' which is the result of the current iteration performed on image L. Image F' is EXCLUSIVE-ORed with image L to detect any differences in the images. If no differences are detected, image F' is the skeleton of image L and will be gated out of the machine as a new image is clocked into latch A. If a difference is detected, image F' is gated into the input of latch A for additional processing. Latches A, B, C, B' and C' are then clocked to store new images in them, and processing continues as described above. At this point latch B' contains image J, latch B contains image J^* which is the result of the first subfield operation on image J, latch C' contains image K, latch C contains image K^{**} which is the result of the second subfield operation on image K, and latch A contains either a new image or the result of the last subfield operation on image L. Successive subfield operations continue on each of the images until no reduction of the image is obtained in one complete iteration, and the skeleton of the image is gated out of the machine. When three Golay neighbor planes generators are utilized to obtain higher data processing rates, the serial procedure for generating composite Golay neighbor planes is not required, and any type of index recognition circuit can be employed.

Figure 6.6 is a schematic of the minimum hardware, modified pipeline implementation of the skeletonizing algorithm. The timing diagram provided in Figure 6.7 shows that the cycle time of this machine is 144 gate delays. For a particular image, each iteration of the skeletonizing algorithm requires 432 gate delays. However, because of

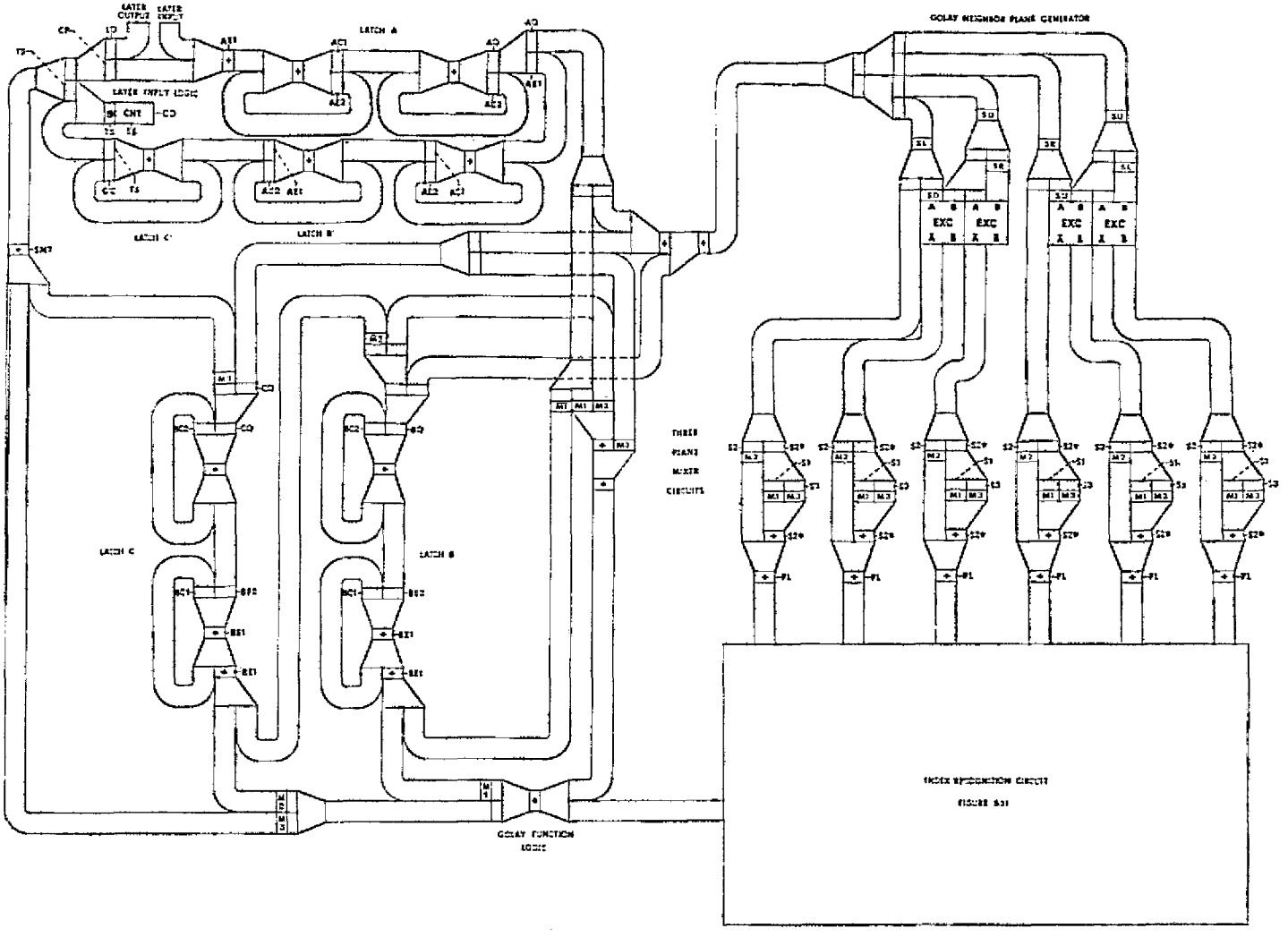


Figure 6.6 Schematic for the minimum hardware, modified pipeline implementation of the skeletonizing algorithm.

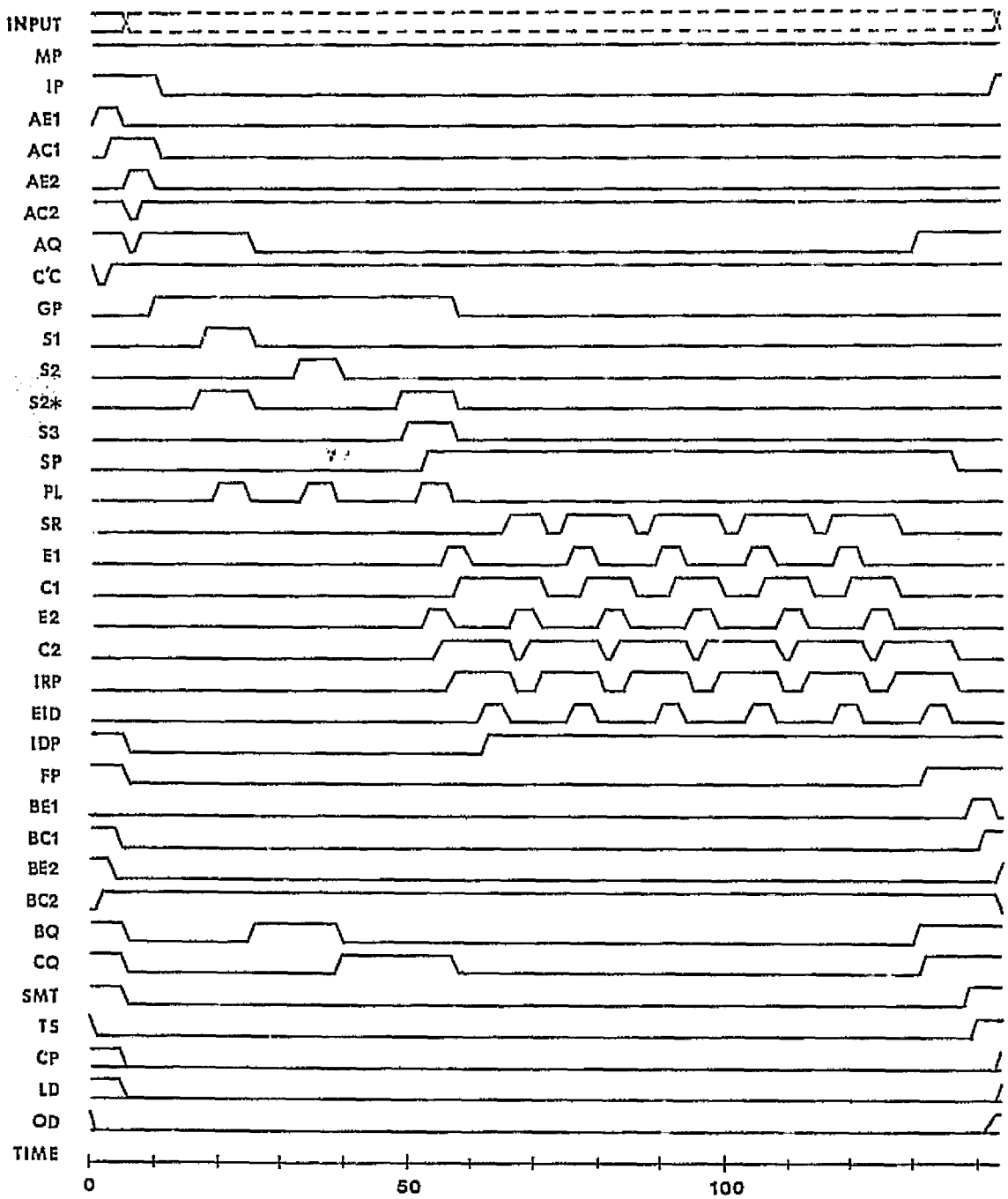


Figure 6.7 Timing diagram for the minimum hardware, modified pipeline implementation of the skeletonizing algorithm.

the pipeline organization and simultaneous processing of three different images, the effective or average time for each iteration is only 144 gate delays. As a result, this machine can process 83 simple images per minute compared to only 38 simple images per minute for the machines described in Chapter 5 which use the same index recognition circuit.

Thirty-four control signals are required by this machine. Their functions are outlined in Table 6.1. The hardware cost function for this implementation of the skeletonizing algorithm is 183A + 129B for a total of 312 components. A measure of the efficiency of this design is provided by the speed-power product of 120 watt-seconds compared to over 229 watt-seconds for the hardwired machines described in Chapter 5.

Additional Modified Pipeline Implementations of the Skeletonizing Algorithm

Figure 6.8 illustrates the general schematic for modified pipeline implementations of the skeletonizing algorithm when three Golay neighbor planes generators are available. The index recognition circuit can be of any desired type. Figure 6.9 is the timing diagram for the case of the shift register based index recognition circuit which was utilized in the minimum hardware, modified pipeline skeletonizing machine. Machine cycle time is reduced to 108 gate delays for a processing rate of 111 simple images per minute. The hardware cost function is 193A + 157B for a total of 350 components. Elimination of the serial procedure for generating the composite Golay neighbor planes reduces the speed-power product by 14.5 percent to 102 watt-seconds.

TABLE 6.1

MINIMUM HARDWARE, MODIFIED PIPELINE SKELETONIZING
MACHINE CONTROL SIGNALS

Control Signal	Number of Conponents Controlled	Control Function
MP	5	Machine Power
IP	3	Input Logic Power
AE1	3	Layer Input Gate
AC1	2	Latch A Slave
AE2	2	Latch A Slave
AC2	2	Latch A Master
AQ	2	Latch A Output
C'C	1	Latch C' Output
GP	23	GNPG Power
S1	6	Mixer Subfield 1
S2	6	Mixer Subfield 2
S2*	12	Mixer Subfield 2 NOT
S3	6	Mixer Subfield 3
SP	18	Shift Register Power
PL	6	Shift Register Input
SR	6	Shift Slave into Master
E1	6	Shift Right to Next Latch
C1	6	Store Slave
E2	6	Input to Master
C2	6	Store Master
IRP	16	Index Recognition Combinational Logic Power
EID	1	Index Recognition Latch Inout
IDP	2	Index Recognition Latch Power
FP	17	Golay Function and Multiplexer Power
BE1	4	Latch B and C Slave Input
BC1	2	Latch B and C Slave Feedback Path
BE2	2	Latch B and C Master Input
BC2	2	Latch B and C Master Feedback Path
BQ	1	Latch B Output
CQ	2	Latch C Output
SMT	1	Subfield Multiplexer Output
TS	4	Contractor Power
CP	1	Continue Processing Control
LD	1	Layer Output Control
OD	0	Contractor Output Signal

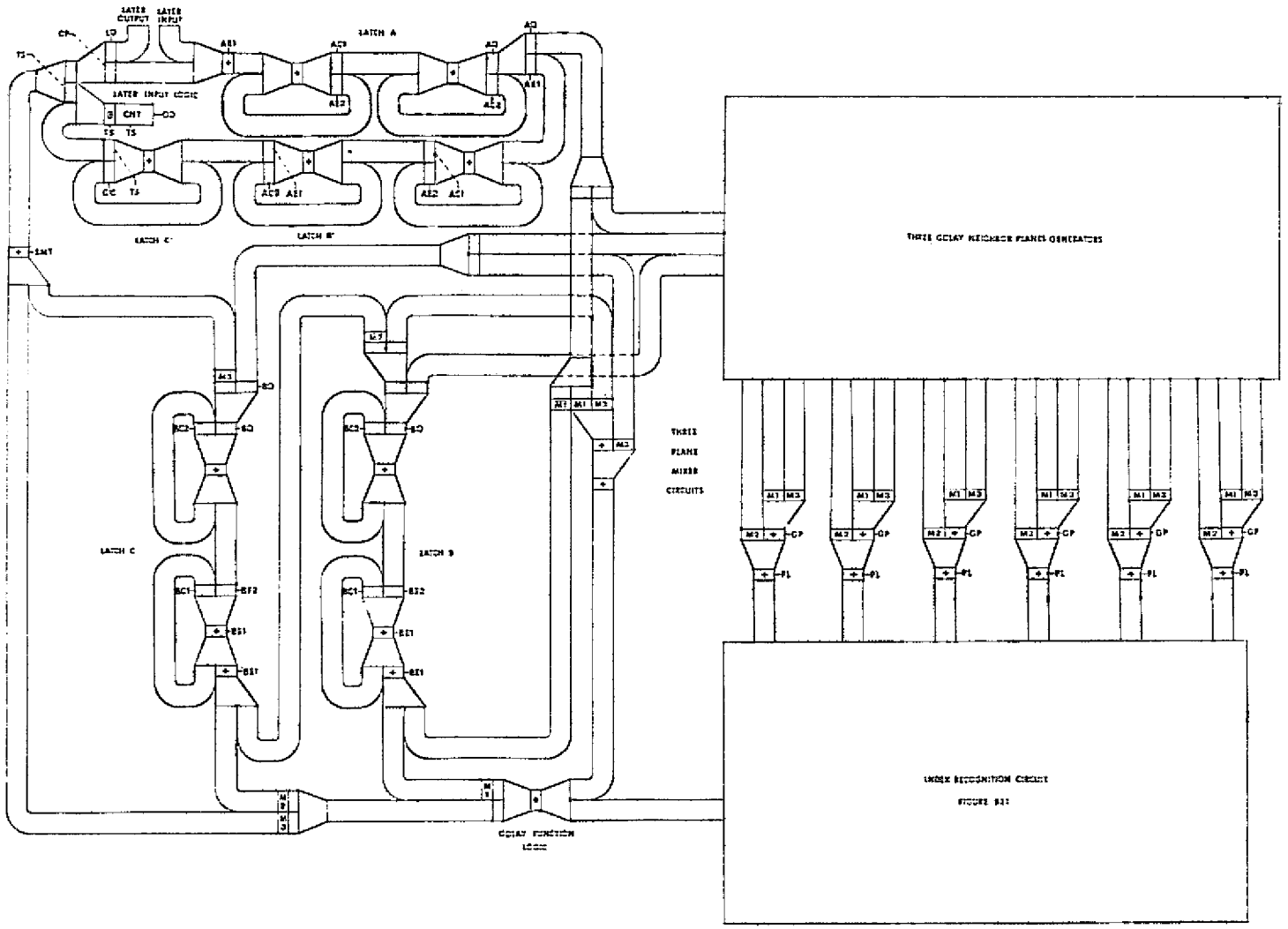


Figure 6.8 Schematic for the modified pipeline implementation of the skeletonizing algorithm with three Golay neighbor planes generators.

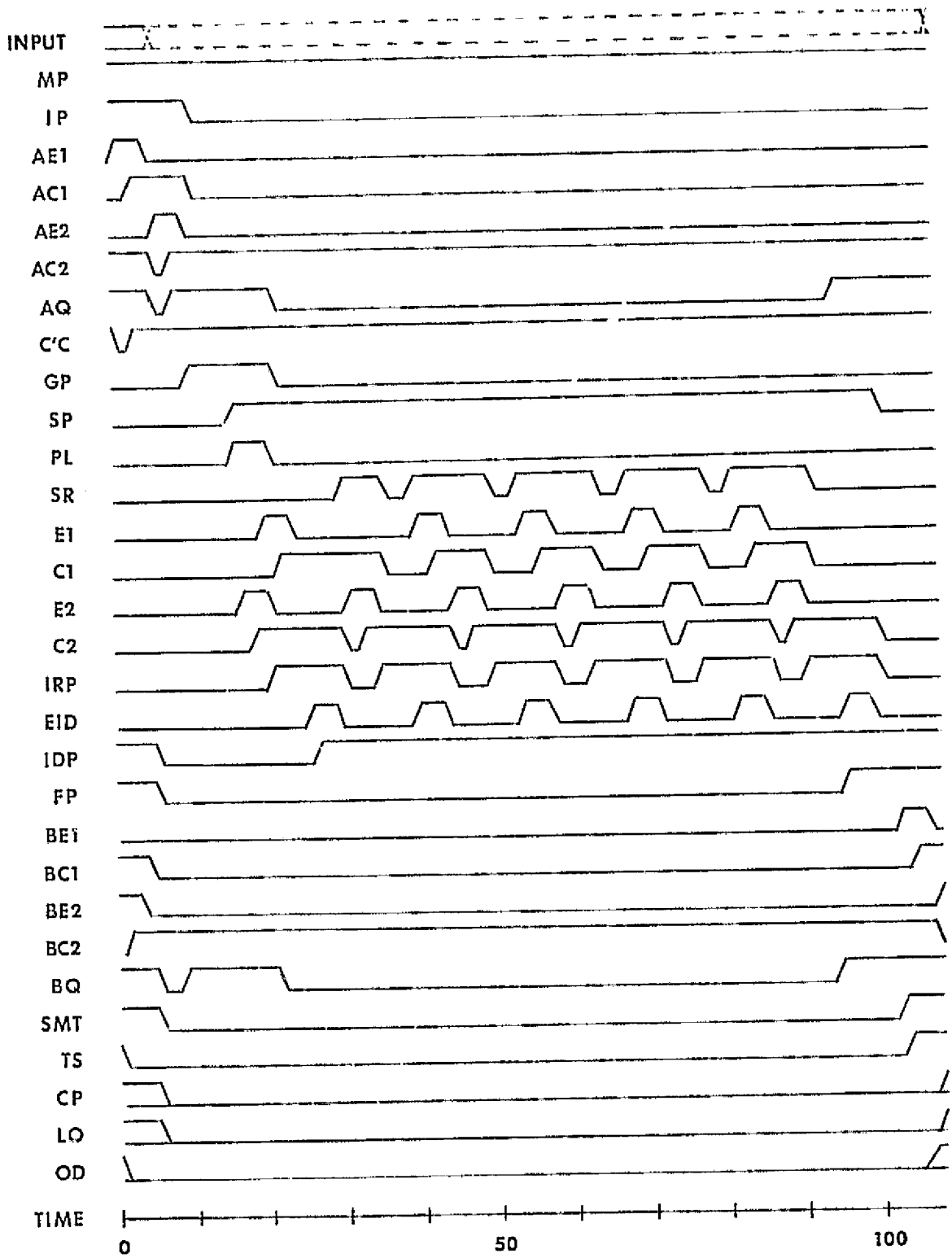


Figure 6.9 Timing diagram for the modified pipeline skeletonizing machine with the shift register based index recognition circuit.

This implementation of the skeletonizing machine provides approximately twice the data processing capability of the OR latch machine presented in Chapter 5 while reducing the speed-power product by 56 percent. The cost of this improved performance is a 57 percent increase in the number of active components and a 76 percent increase in the number of passive components required to build the skeletonizing machine. The functions of the 29 control signals required by this machine are outlined in Table 6.2.

For applications which require ultra high data processing rates, a space iterative index recognition circuit can be used in the skeletonizing machine in Figure 6.8. A timing diagram for this implementation of the skeletonizing algorithm is illustrated in Figure 6.10. Control signal functions are described in Table 6.3. The hardware cost of this machine is 249A + 189B. A total of 438 components, or more than twice the number of components required by the OR latch skeletonizing machine, are utilized in this ultrahigh speed design. Figure 6.10 shows that the machine cycle time is only 34 gate delays which allows the skeletonizing machine to process 353 simple images per minute. This is approximately nine times the processing speed of the OR latch machine described in Chapter 5. The speed-power product of this high speed, modified pipeline skeletonizing machine is 93 watt-seconds versus 229 watt-seconds for the OR latch machine. The characteristics of the three modified pipeline skeletonizing machines described in this chapter are summarized in Table 6.4.

Extension of the modified pipeline organization described in this chapter to the realization of Golay transforms involving four or seven

TABLE 6.2

CONTROL SIGNALS FOR THE MODIFIED PIPELINE SKELETONIZING MACHINE WITH THREE GOLAY NEIGHBOR PLANES GENERATORS

Control Signals	Number of Components Controlled	Control Function
MP	5	Machine Power
IP	3	Input Logic Power
AE1	3	Layer Input Gate
AC1	2	Latch A Slave
AE2	2	Latch A Slave
AC2	2	Latch A Master
AQ	2	Latch A Output
C'C	1	Latch C' Output
GP	63	GNPG Power
SP	18	Shift Register Power
PL	6	Shift Register Input
SR	6	Shift Slave into Master
E1	6	Shift Right to Next Latch
C1	6	Store Slave
E2	6	Input to Master
C2	6	Store Master
IRP	16	Index Recognition Combinational Logic Power
EID	1	Index Recognition Latch Input
IDP	2	Index Recognition Latch Power
FP	17	Golay Function and Multiplexer Power
BE1	4	Latch B and C Slave Input
BC1	2	Latch B and C Slave Feedback Path
BE2	2	Latch B and C Master Input
BC2	2	Latch B and C Master Feedback Path
BQ	3	Latch B Output
SMT	1	Subfield Multiplexer Output
TS	4	Contractor Power
CP	1	Continue Processing Control
LO	1	Layer Output Control
OD	0	Contractor Output Signal

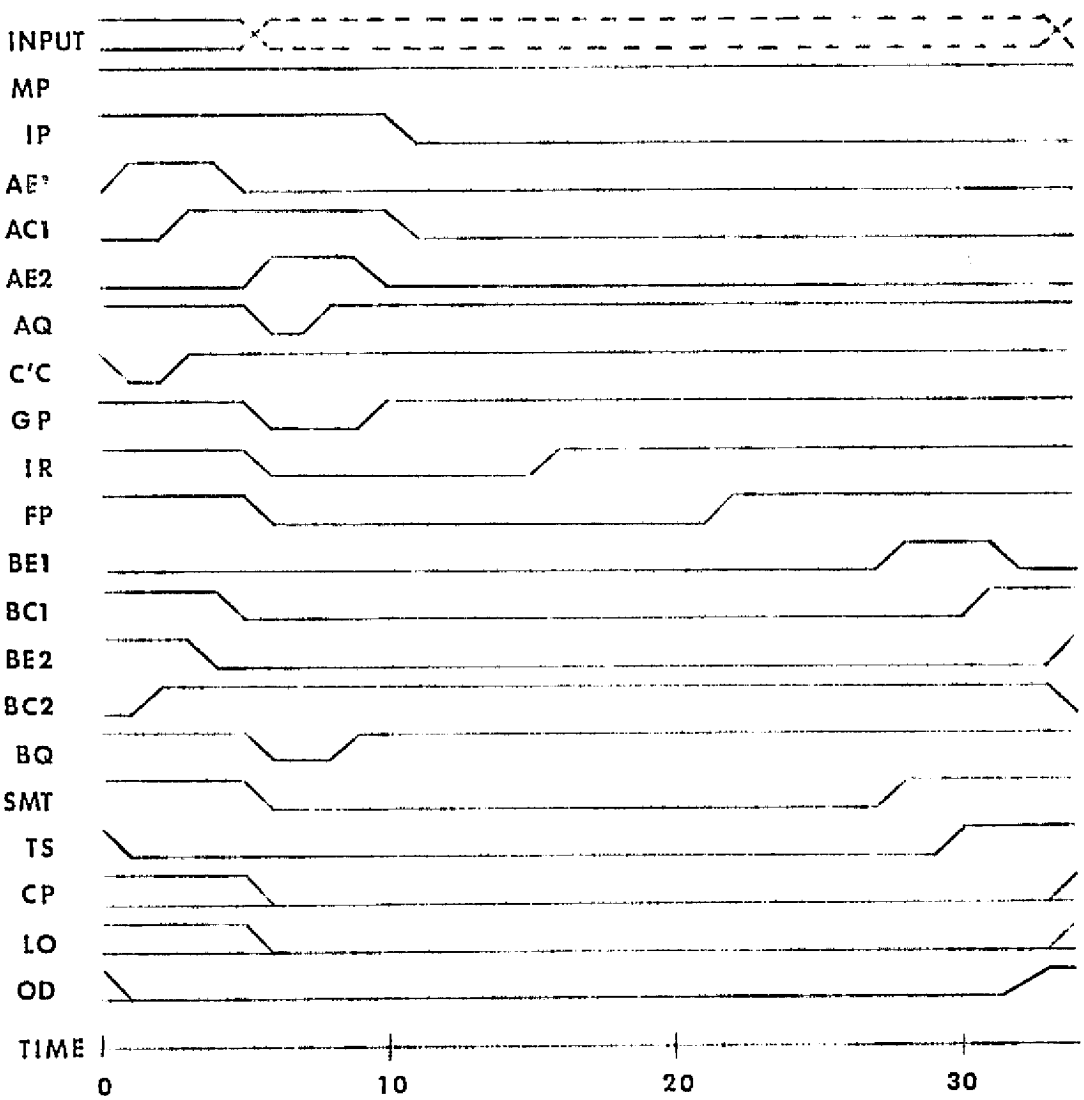


Figure 6.10 Timing diagram for the modified pipeline skeletonizing machine with the space iterative index recognition circuit.

TABLE 6.3

CONTROL SIGNALS FOR THE MODIFIED PIPELINE SKELETONIZING MACHINE WITH THE SPACE ITERATIVE INDEX RECOGNITION CIRCUIT

Control Signal	Number of Components Controlled	Control Function
MP	5	Machine Power
IP	3	Input Logic Power
AE1	3	Layer Input Gate
AC1	2	Latch A Slave
AE2	2	Latch A Slave
AQ	4	Latch A Output
C'C	1	Latch C' Output
GP	63	GNPG Power
IR	131	Index Recognition Circuit Power
FP	15	Golay Function and Multiplexer Power
BE1	4	Latch B and C Slave Input
BC1	2	Latch B and C Slave Feedback Path
BE2	2	Latch B and C Master Input
BC2	2	Latch B and C Master Feedback Path
BQ	3	Latch B Output
SMT	1	Subfield Multiplexer Output
TS	4	Contractor Power
CP	1	Continue Processing Control
LO	1	Layer Output Control
OD	0	Contractor Output Signal

TABLE 6.4

PERFORMANCE CHARACTERISTICS OF THE MODIFIED PIPELINE SKELETONIZING MACHINES

Machine Type	Cost Function	Number of Control Signals	Total Component Count	Gate Delays per Iteration (Time in Seconds)	Data Rate Simple Images per Minute	Average Power Consumption in Watts	Peak Power Consumption in Watts	Speed-Power Product in Watt-Seconds
MINIMUM HARDWARE MODIFIED PIPELINE	183A+129B	34	312	144 (0.72)	83.33	166.19	255	119.66
MODIFIED PIPELINE WITH SHIFT REGISTER INDEX RECOGNITION	193A+157B	29	350	108 (0.54)	111.11	189.50	363	102.33
MODIFIED PIPELINE WITH SPACE ITERATIVE INDEX RECOGNITION	249A+189B	19	438	34 (0.17)	352.94	545.12	714	92.67

subfields is straightforward. Each additional subfield requires two additional latches, one for storage of the input image and one for temporary storage of the additional partial result from the subfield operation for that subfield. The three plane mixer circuits become four or seven plane mixers which consist of a mask for each subfield and OR gates to combine the multiple masked inputs into a single composite output image. Four or seven Golay neighbor planes generators are required unless the sequential technique for generating the composite Golay neighbor planes is employed. The two plane mixer circuits require only a mask change to insure that the correct subfields from the two input images are combined to form the composite output image. Since the Golay neighbor planes generator for the four subfields case is much simpler than for the three or seven subfields case, the case of four Golay neighbor planes generators does not increase the basic hardware cost of the Golay transform machine. However, the seven Golay neighbor planes generators for the seven subfields case represent a substantial increase in hardware cost for the basic Golay transform machine organization.

This chapter presented the development of modified pipeline realizations of the skeletonizing algorithm which are useful in applications that require high data processing rates. Chapter 7 will describe the design of a programmable tse computer architecture which is capable of performing Golay transforms with a relatively small number of tse logic devices.

CHAPTER 7

A SPECIAL PURPOSE PROGRAMMABLE TSE PROCESSOR

A number of different architectures for hardwired tse logic skeletonizing machines have been presented in previous chapters. These machines have the advantage of providing relatively high data processing rates but do not offer the flexibility which could be obtained with a programmable tse processor. This chapter presents a special purpose programmable tse computer which can be used to perform numerous Golay transforms. A microcomputer-based, tse computer control unit is described and used to define a basic instruction set for the tse processor. Programs for performing the Golay transform skeletonizing and swelling algorithms are illustrated. In addition, the use of advanced microprogramming techniques to define additional instructions for the tse computer is demonstrated.

A Programmable Tse Computer

Figure 7.1 is a block diagram of a special purpose tse computer which is designed to perform Golay transform operations on images divided into three subfields. The machine consists of an arithmetic-logic unit (ALU), an accumulator latch (A), two general purpose latches (B and C), an index recognition latch (I), an output latch (O), a contractor, an index recognition circuit, and a control unit. The ALU includes a latch which temporarily stores the result of an ALU operation when the accumulator or a general purpose latch is being used as both a source and a destination register for the current

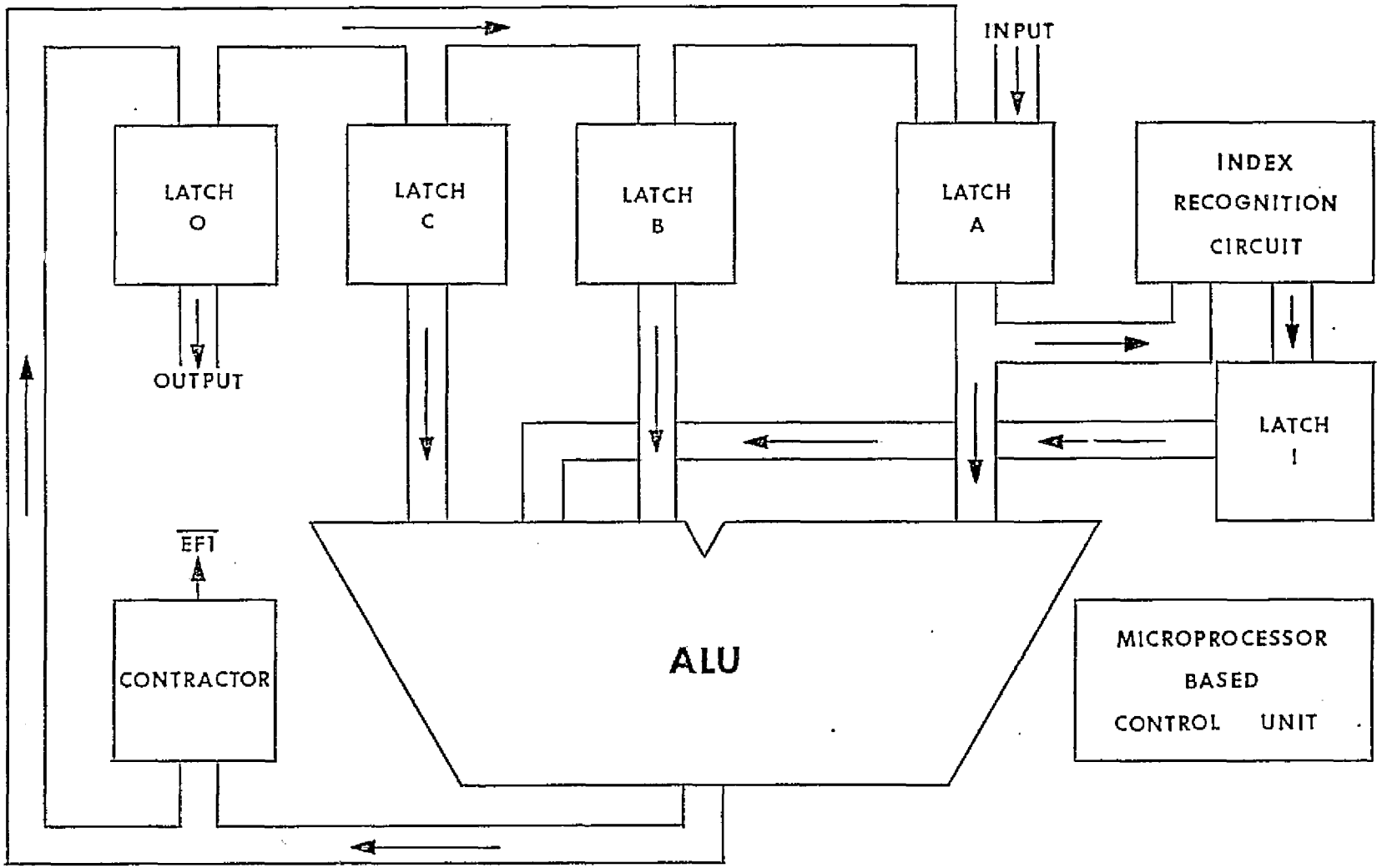


Figure 7.1 A special purpose tse computer organization.

operation. This prevents undesirable race conditions from developing in the tse processor.

The accumulator and general purpose registers, B and C, serve as both source and destination registers. Latch O can only function as a destination register, and latch I can only function as a source register for ALU operations. In the ALU operations which require two operands, the accumulator is entered in the right side of the ALU as one operand, while the output of latch B, C or I is entered in the left side of the ALU as the second operand. The ALU is capable of performing the AND, EXCLUSIVE-OR, and COMPLEMENT operations. Images can also be gated through the ALU to the input of any destination register. Since the registers are OR latches, the OR operation is normally performed by gating one operand through the ALU and ORing that image with the second operand which is stored in the destination register.

Two independent mask generator circuits of the type illustrated in Figure 7.2 are included in the ALU. An image mask for any of the three possible subfields or any combination of those subfields can be created by controlling the states of the three tse read-only-memories. For example, if M1, M2, and M are all active, the output image will be M3. The ALU is organized so that these mask tses can be ORed or ANDed with the ALU input images from the source registers. This provides the capability of performing ALU operations on entire images or only on selected subfields of the images. The result of each ALU operation is tested by the contractor which detects all zero tses.

In addition to the ALU, the tse computer utilizes an index

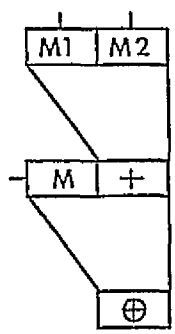


Figure 7.2 A three subfield mask generator circuit.

recognition network consisting of a Golay neighbor planes generator and a comparison type index recognition circuit. This function could be implemented as a programmed ALU operation using individual logical and slide operations but would then require extremely long execution times. The hardwired comparison type index recognition circuit provides an effective trade off between the hardware cost and the speed of the Golay transform tse computer while preserving the ability to recognize all fourteen possible indices. The accumulator is the source register for all index recognition operations. Latch I is the destination register.

A schematic of the tse logic used in the Golay transform computer is illustrated in Figure 7.3. The hardware cost function for this machine is 97A+69B. No random access tse memory is included in the computer organization because of the high hardware cost of tse memory. Also, due to the flexibility of the tse logic ALU, external memory requirements should be minimal. In most applications, only a serial input image buffer memory would be required to synchronize the variable length Golay image processing operation to the incoming data. Additional general purpose registers could be added to the tse computer organization if they are needed for temporary storage of partial results from a complex transform operation. This would require less hardware than adding external memory to the tse computer.

Tse Computer Control Unit

Theoretically, a computer control unit should be able to produce the optimum control bit sequence for performing any operation

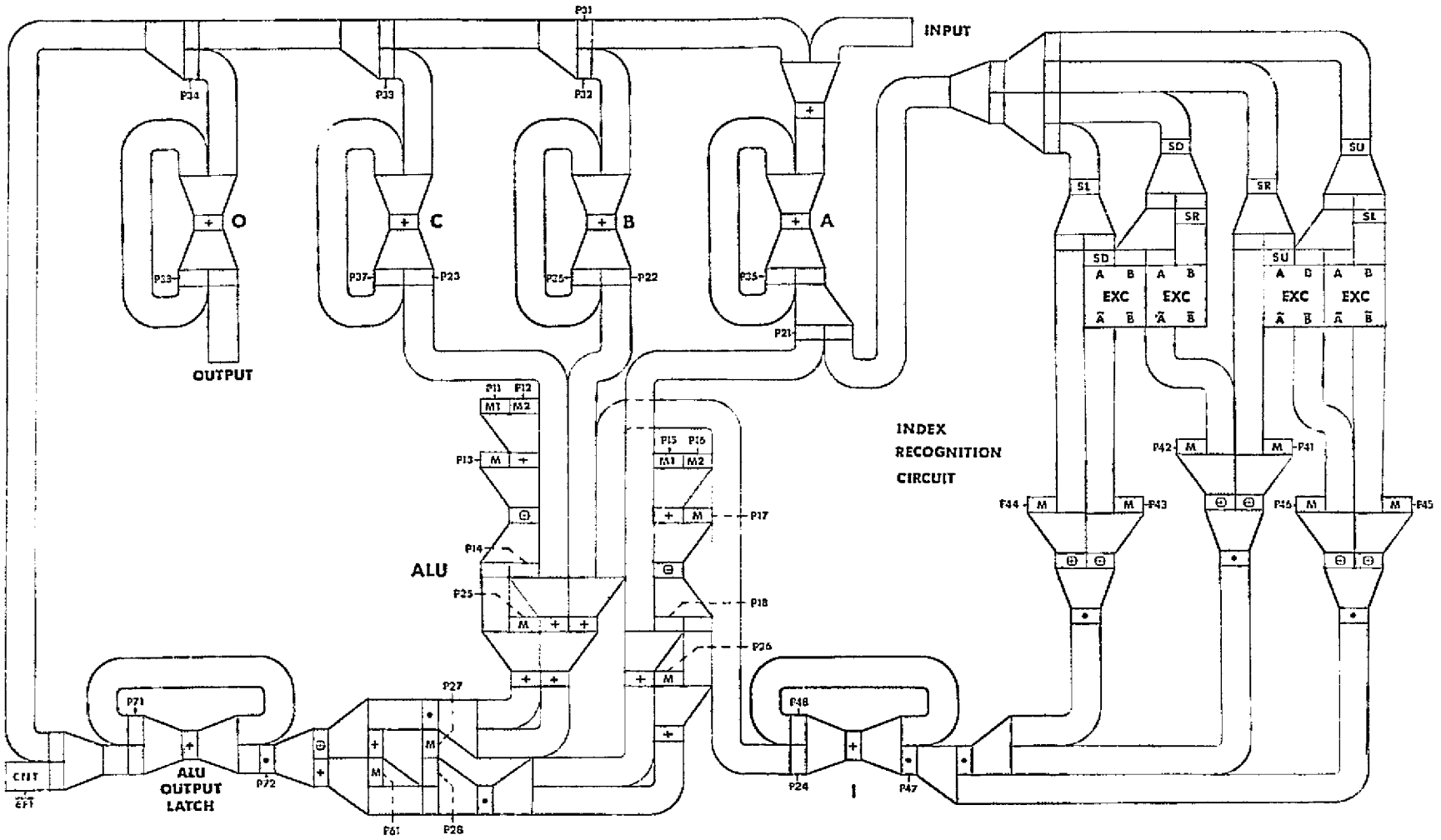


Figure 7.3 Tse logic for the Golay transform tse computer.

which is feasible with the register and ALU organization of the controlled computer. In practice, this degree of flexibility can only be approached by utilizing a microprogrammed control unit which contains the control bit combinations in microinstructions that are read from a control memory. Groups of microinstructions form microprograms that control the execution of each macroinstruction. Thus, microprogrammed computers require a small computing section within the central processing unit (CPU) to execute the microprograms. A block diagram of the organization of a typical microprogrammed control unit is shown in Figure 7.4. The cycle time of a microprogrammed control unit must be significantly faster than the minimum cycle time of the main computer for the microprogrammed control unit to be efficient. In conventional computers this requirement places severe restrictions on the design of the control unit. The trade off between the complexity of the control unit and the time required to decode and execute each instruction must be carefully considered in the design. Since the projected propagation delay of tse logic devices is several orders of magnitude longer than the propagation delay of standard bipolar and MOS logic, a complex conventional logic control unit can potentially provide efficient control of a tse logic ALU.

A tse computer control unit that provides the benefits of microprogramming and permits the use of conventional semiconductor memory for tse computer program storage has been designed. The control unit is based on RCA's CDP 1802 COSMAC microprocessor [19]. Figure 7.5 is a detailed block diagram of the control unit. The 1802 microprocessor was chosen over other currently available devices because of several unique architectural features which enhance the input-output (I/O) and control

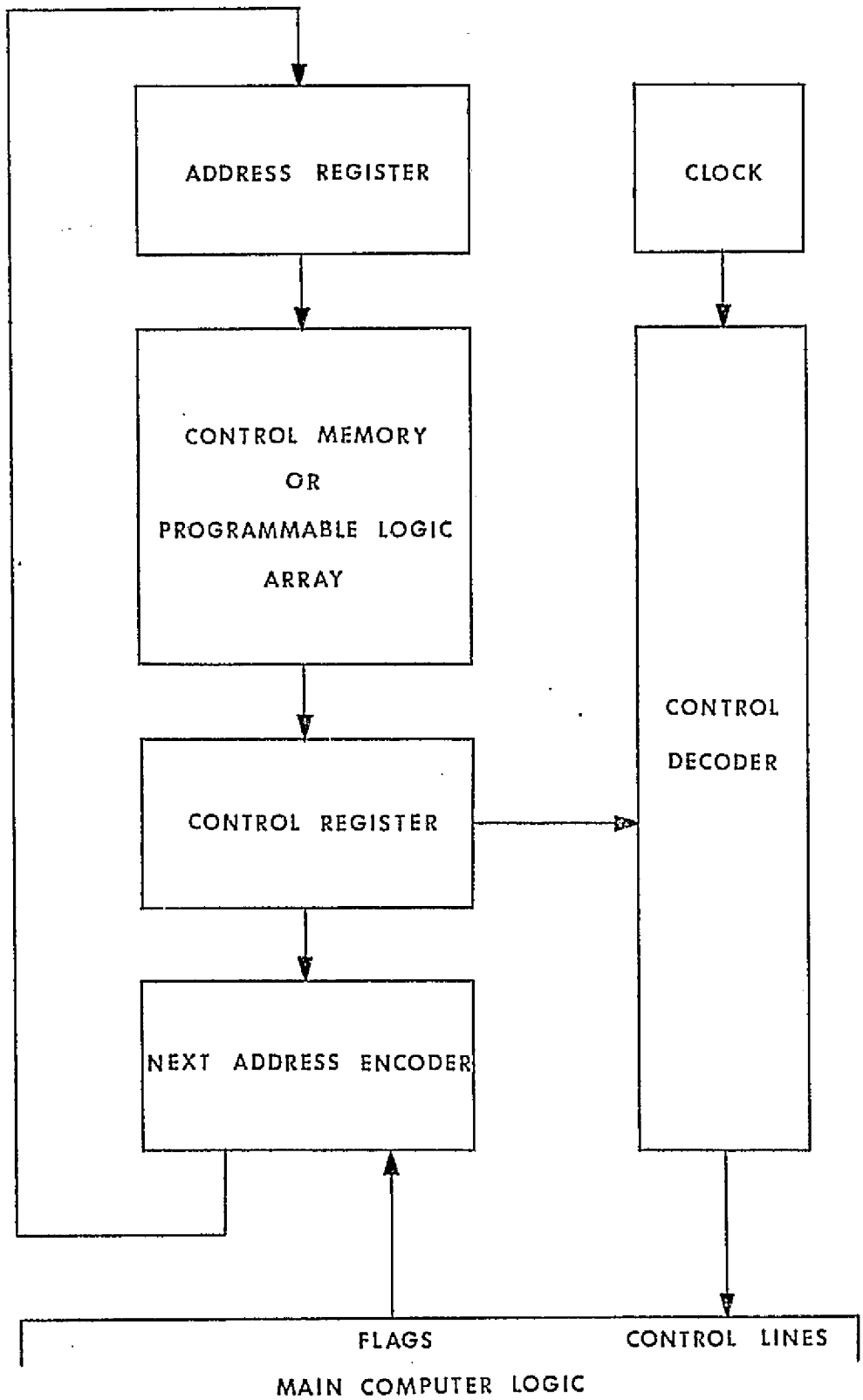


Figure 7.4 Organization of a conventional microprogrammed control unit.

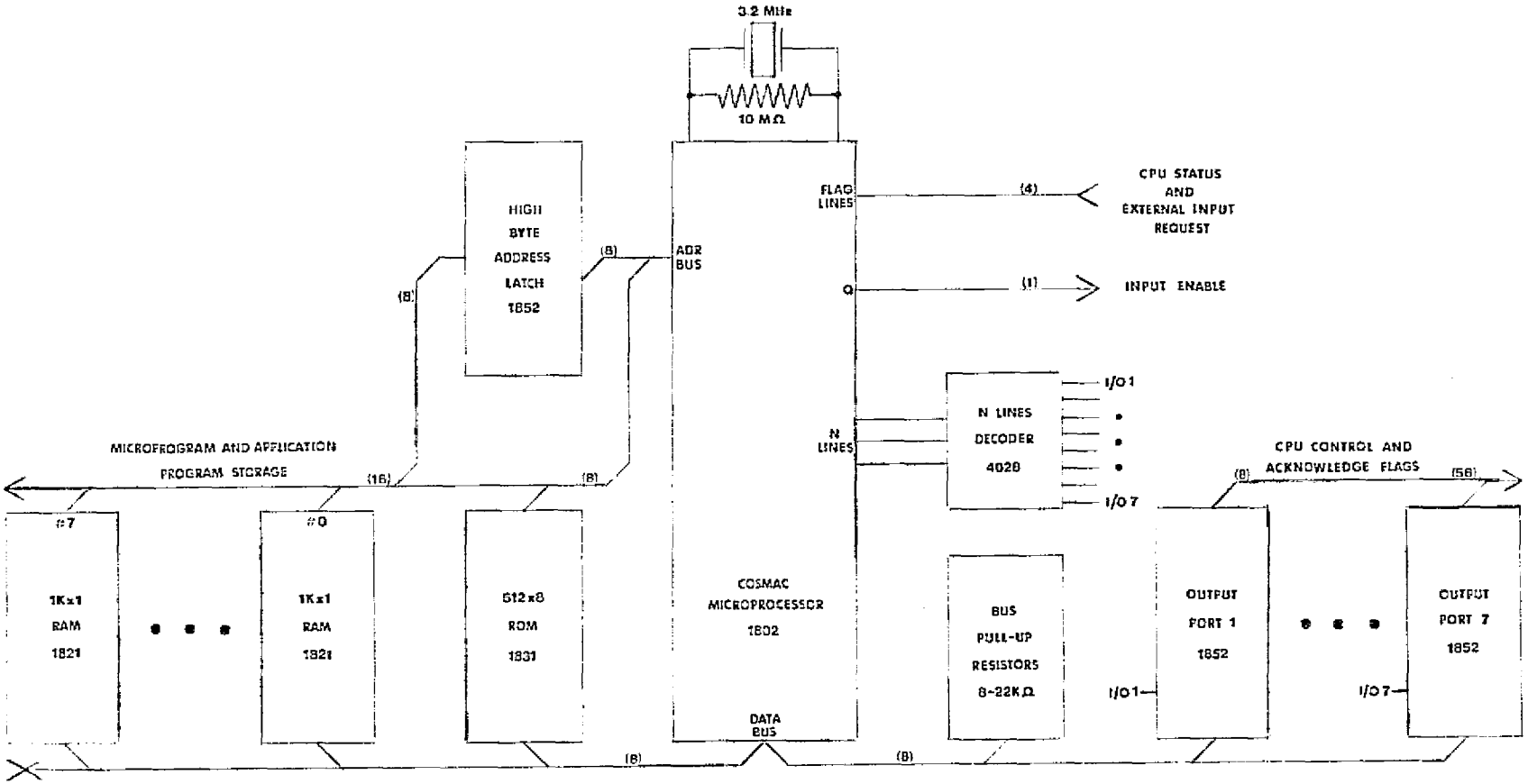


Figure 7.5 Block diagram of the tse computer control unit.

capabilities of the device. These features include on-chip I/O and the use of multiple program counters. A brief summary of the architecture and instruction set of the 1802 is provided in Appendix C.

The 1802 microprocessor controls the tse components by outputting control bytes and executing internal timing loops to account for the propagation delays of the tse logic devices. Flag input EF1 monitors the output of the contractor to detect all zero tses at the output of the ALU. The EF2 flag is used for external input requests. These requests are acknowledged by setting bit three of output port five. Input images are gated into latch A under control of the Q line. At the end of the execution cycle for major tse instructions, EF3 is tested for an external request to halt tse processing. This feature can be used to single step the tse computer through the more complex tse instructions. Conventional techniques [19] can be used to single step the 1802 microprocessor through the simpler tse instructions and the microprograms themselves. Table 7.1 summarizes the function of each major tse control signal.

Tse Computer Instruction Set

An instruction set consisting of all the standard CDP 1802 instructions plus 26 generic tse instructions with over 1300 variations has been developed for the special purpose Golay transform tse computer. The 1802 instruction set is given in Table C.1, Appendix C. COSMAC instructions can be used alternately as macroinstructions or as microinstructions. Table 7.2 summarizes the basic tse instruction

TABLE 7.1

TSE COMPUTER CONTROL SIGNAL FUNCTIONS

Name	Number of Bits	Function
EF1	1	Detects All Zero Tses At ALU Output
EF2	1	External Input Request
EF3	1	External Request to Halt Tse Processing
Q	1	Controls External Input to Latch A
Output Port 1	8	ALU Control
Output Port 2	8	Source Register Output and ALU Control
Output Port 3	8	Destination Register Input and Feedback Control
Output Port 4	8	Index Recognition Circuit Control
Output Port 5	1	Acknowledge Input Request
Output Port 6	1	ALU Control
Output Port 7	2	ALU Output Latch Control

TABLE 7.2

TSE INSTRUCTIONS

Instruction	Mnemonic	Operation
REGISTER OPERATIONS		
MOVE REGISTER TO REGISTER (SEE TOUT)	TMOV REG1,REG2	(REG2)→REG1
MOVE IMMEDIATE TO REGISTER	TMVI REG,MASK	(MASK)→REG
CLEAR REGISTER I	TCLRI	0→I
LOGIC OPERATIONS		
AND REGISTER TO A	TAND REG1, REG2, MASK	$[(\overline{MASK})+(REG2)]\cdot(A)\rightarrow REG1$
AND A TO REGISTER	TANA REG1,REG2,MASK	$[(\overline{MASK})+(A)]\cdot(REG2)\rightarrow REG1$
AND IMMEDIATE TO REGISTER	TANI REG1,REG2,MASK	$(MASK)\cdot(REG2)\rightarrow REG1$
AND $\overline{\text{REGISTER}}$ TO A	TANDN REG1,REG2,MASK	$[\overline{(MASK)\cdot(REG2)}]\cdot(A)\rightarrow REG1$
AND $\overline{A}$ TO REGISTER	TANAN REG1,REG2,MASK	$[\overline{(MASK)\cdot(A)}]\cdot(REG2)\rightarrow REG1$
OR REGISTER TO A	TOR REG,MASK	$[(MASK)\cdot(REG)]+(A)\rightarrow A$
OR A TO REGISTER	TORA REG,MASK	$[(MASK)\cdot(A)]+(REG)\rightarrow REG$
OR IMMEDIATE TO REGISTER	TORI REG,MASK	$(MASK)+(REG)\rightarrow REG$
OR $\overline{\text{REGISTER}}$ TO A	TORN REG,MASK	$\overline{[(MASK)+(REG)]}+(A)\rightarrow A$
OR $\overline{A}$ TO REGISTER	TORAN REG, MASK	$\overline{[(MASK)+(A)]}+(REG)\rightarrow REG$
EXCLUSIVE-OR REGISTER TO A	TXOR REG1,REG2,MASK	$[(MASK)\cdot(REG2)]\oplus(A)\rightarrow REG1$
EXCLUSIVE-OR A TO REGISTER	TXRA REG1,REG2,MASK	$[(MASK)\cdot(A)]\oplus(REG2)\rightarrow REG1$
EXCLUSIVE-OR IMMEDIATE TO REGISTER COMPLEMENT REGISTER	TXRI REG1,REG2,MASK TCMR REG1,REG2,MASK	$(MASK)\oplus(REG2)\rightarrow REG1$

TABLE 7.2 (continued)

Instruction	Mnemonic	Operation
INDEX RECOGNITION		
IDENTIFY BASIS POINTS WITH A SURROUND OF INDEX X	TIDA X	(I)+(ID)→I Where ID is a tse with 1's in positions which correspond to basis points of A which have a surround of index X.
COMPARE OPERATIONS		
CONTRACT REGISTER	TCNT REG,MASK	(MASK)·(REG);IF=0, 0→RF IF≠0, 1→RF
TEST REGISTER	TTEST REG,MASK	$(\overline{MASK})$+(REG);IF=0, 0→RF IF≠0, 1→RF
COMPARE REGISTER	TCMP REG,MASK	[(MASK)·(REG)]⊕[(MASK)·(A)]; IF RESULT=0, 0→RF IF RESULT≠0, 1→RF
COMPARE IMMEDIATE	TCPI REG,MASK	(MASK)⊕(REG); IF(REG)=(MASK), 0→RF IF(REG)≠(MASK), 1→RF
TSE BRANCH OPERATIONS		
TSE SHORT BRANCH ON ZERO	TBZ ADR	IF RF=0,M(R(P))→R(P).0 ELSE R(P)+1
TSE SHORT BRANCH ON NO ZERO	TBNZ ADR	IF RF≠0,M(R(P))→R(P).0 ELSE R(P)+1
TSE LONG BRANCH ON ZERO	TLBZ ADR	IF RF=0,M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
TSE LONG BRANCH ON NO ZERO	TLBNZ ADR	IF RF=1,M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
INPUT/OUTPUT OPERATIONS		
INPUT TSE	TIN	INPUT TSE→A
OUTPUT TSE (SEE TMOV)	TOUT REG	(REG)→O

set which was created by using the 1802 instructions as microinstructions. The dummy arguments REG, REG1, REG2, and MASK should be replaced by the appropriate arguments from Table 7.3 when programs are written using the COSMAC or tse instructions.

Tse instructions are divided into six basic groups consisting of register operations, logic operations, index recognition operations, compare operations, branch operations, and input-output operations. The register operations facilitate movement of tse data within the computer. As is the case in all tse instructions, register 0 cannot be specified as a source register, and register I cannot be specified as a destination register. Logic instructions provide the operations which are necessary to perform Golay functions. In general, register A cannot be specified as REG2 when register A is an implicit operand in the logic operations. The contents of any source register can be tested using the compare operations. These instructions are typically used to determine whether or not an image was altered by the last iteration of a Golay transform. The branch instructions provide a method for testing the results of a tse operation and for performing conditional operations. Both short branches, which are limited to the current memory page, and long branches, which can specify any memory location, are included in the instruction set. Note that the tse branch instructions depend on the contents of R(F).0 which is set to one or zero during each compare operation. The standard 1802 instructions provide additional branching capabilites. For example, the B2 and BN2 instructions are used to test for external input requests.

TABLE 7.3

REGISTER AND MASK CONSTANT DEFINITIONS

Name	Symbol	Decimal Value
COSMAC REGISTERS		
REGISTER 0	R0	0
REGISTER 1	R1	1
REGISTER 2	R2	2
REGISTER 3	R3	3
REGISTER 4	R4	4
REGISTER 5	R5	5
REGISTER 6	R6	6
REGISTER 7	R7	7
REGISTER 8	R8	8
REGISTER 9	R9	9
REGISTER 10	RA	10
REGISTER 11	RB	11
REGISTER 12	RC	12
REGISTER 13	RD	13
REGISTER 14	RE	14
REGISTER 15	RF	15
TSE REGISTERS		
REGISTER A	A	225
REGISTER B	B	210
REGISTER C	C	180
REGISTER O	O	120
REGISTER I	I	8
TSE MASKS		
ALL ZERO MASK	M0	0
ALL ONE MASK	M	12
SUBFIELD ONE MASK	M1	9
SUBFIELD TWO MASK	M2	10
SUBFIELD THREE MASK	M3	15
SUBFIELDS ONE AND TWO MASK	M12	11
SUBFIELDS ONE AND THREE MASK	M13	14
SUBFIELDS TWO AND THREE MASK	M23	13

Microprogram Control of Tse Operations

When an application program is assembled for the tse computer, each tse instruction generates a multiple byte operational code which is actually a group of 1802 instructions and data bytes. This information is used to produce the control signal sequence required to perform the specified tse operation. Some tse instructions, such as TCLRI and the tse branch instructions, require less than six bytes of microcode to control their execution. The microcode expansions of these instructions are used directly as their operational codes. Thus, these instructions are self-contained in the sense that they do not require a separate microprogram to control their execution. The remaining tse instructions require somewhat longer control programs which are executed as called subroutines. The subroutine call is performed by an 1802 set P instruction which is used as the first byte of the tse instruction operational code. Control signal sequences are specified by the remaining bytes of the operational code. Control microprograms always return with a SEP R3 instruction since all tse computer applications programs will use R(3) as their program counter.

Figure 7.6 is a flow chart for the general ALU operations control program which is listed in Figure D.1, Appendix D. This microprogram controls the execution of all the tse register and logic operations except TCLRI. These instructions require a six byte operational code. The first byte is a SEP R9 instruction which calls the general ALU operations microprogram. The remaining five bytes contain the ALU mask, tse register output, port six, tse register input number one, and tse register input number two control bytes which are output by the ALU

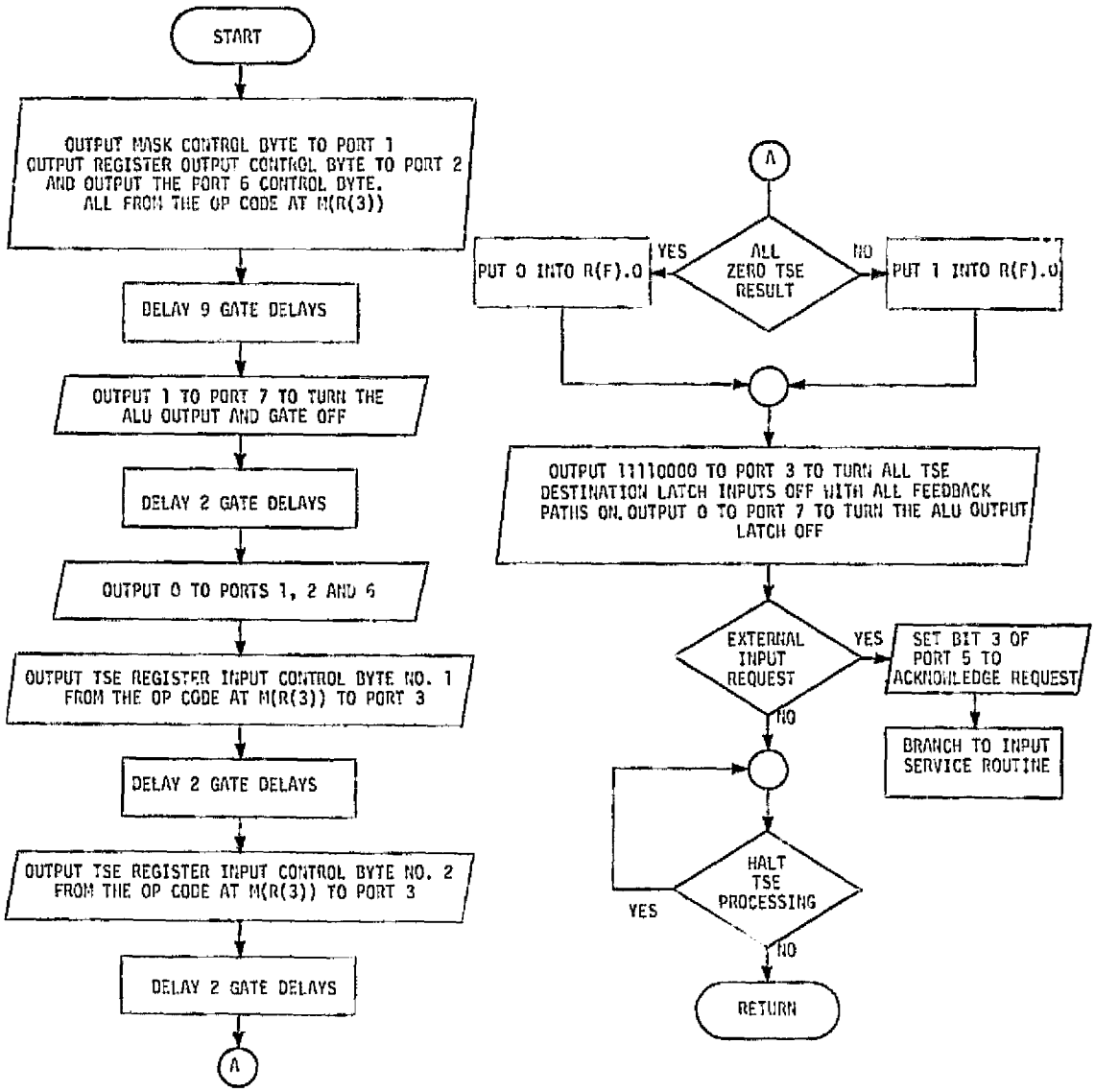


Figure 7.6 A flow chart for the general ALU operations control program.

operations microprogram to control execution of the specified tse instruction.

The general ALU operations control program utilizes the versatile I/O capability of the 1802 to minimize the complexity of the control microprogram. Control bytes which vary from instruction to instruction within the tse register and logic instruction classes are output directly from the application program memory space addressed by R(3). This eliminates the need to specifically decode the individual tse register and logic instructions before initiating their execution. Constant control bytes are output as immediate data from the control microprogram to minimize the length of the tse instruction operational codes. The 1802 microprocessor is particularly efficient at performing these tasks because the output data pointer, R(X), is automatically incremented during each output operation, and the register which is assigned as the data pointer can be changed by a single set X instruction.

Time delays are included in the microprogram to account for the relatively long propagation delay of the tse logic devices. The long delay subroutine MLDLY listed in Figure D.2, Appendix D, is called by the ALU operations microprogram to create the time delays. A standard subroutine call and return technique [19] is employed, and two data bytes are passed to MLDLY to specify the length of the delay. The 1802 executes most COSMAC instructions in two machine cycles which consist of eight machine states each. To simplify control signal timing, a 3.2MHz clock frequency was chosen for the 1802 microprocessor. This frequency permits the 1802 and associated components to be operated

from a five volt power supply and provides exactly 2000 machine states in five milliseconds. Thus, 1000 two-cycle 1802 instructions can be executed within the propagation delay of a tse logic gate.

Figure 7.7 is a timing diagram for the tse register and logic instructions which execute under control of the general ALU operations microprogram. The execution time for these instructions is 19 gate delays. Flags EF1, EF2, and EF3 are tested during the execution of these instructions. At the end of the tse instruction execution cycle, R(F).0 will contain zero if the image at the output of the ALU was an all zero tse. Otherwise, R(F).0 will be set equal to one. Note that the contents of R(F).0 is not necessarily indicative of the result of a tse OR instruction since the final OR operation is normally performed at the destination register.

A listing of the tse computer compare operations control program is provided in Figure D.3, Appendix D, and a flow chart for the microprogram is shown in Figure 7.8. The tse compare instructions have four byte operational codes which consist of a SEP RC instruction and three control bytes. Only three control bytes are required because there is no destination register. With this single exception, the tse compare operations microprogram performs essentially the same function as the general ALU operations microprogram. Tse compare instructions execute in 15 gate delays (Figure 7.9).

One of the most complex tse computer operations is performed by the index recognition instruction. The index recognition control microprogram is listed in Figure D.4, Appendix D. Figure 7.10 is a flow chart for this microprogram. The index recognition instruction

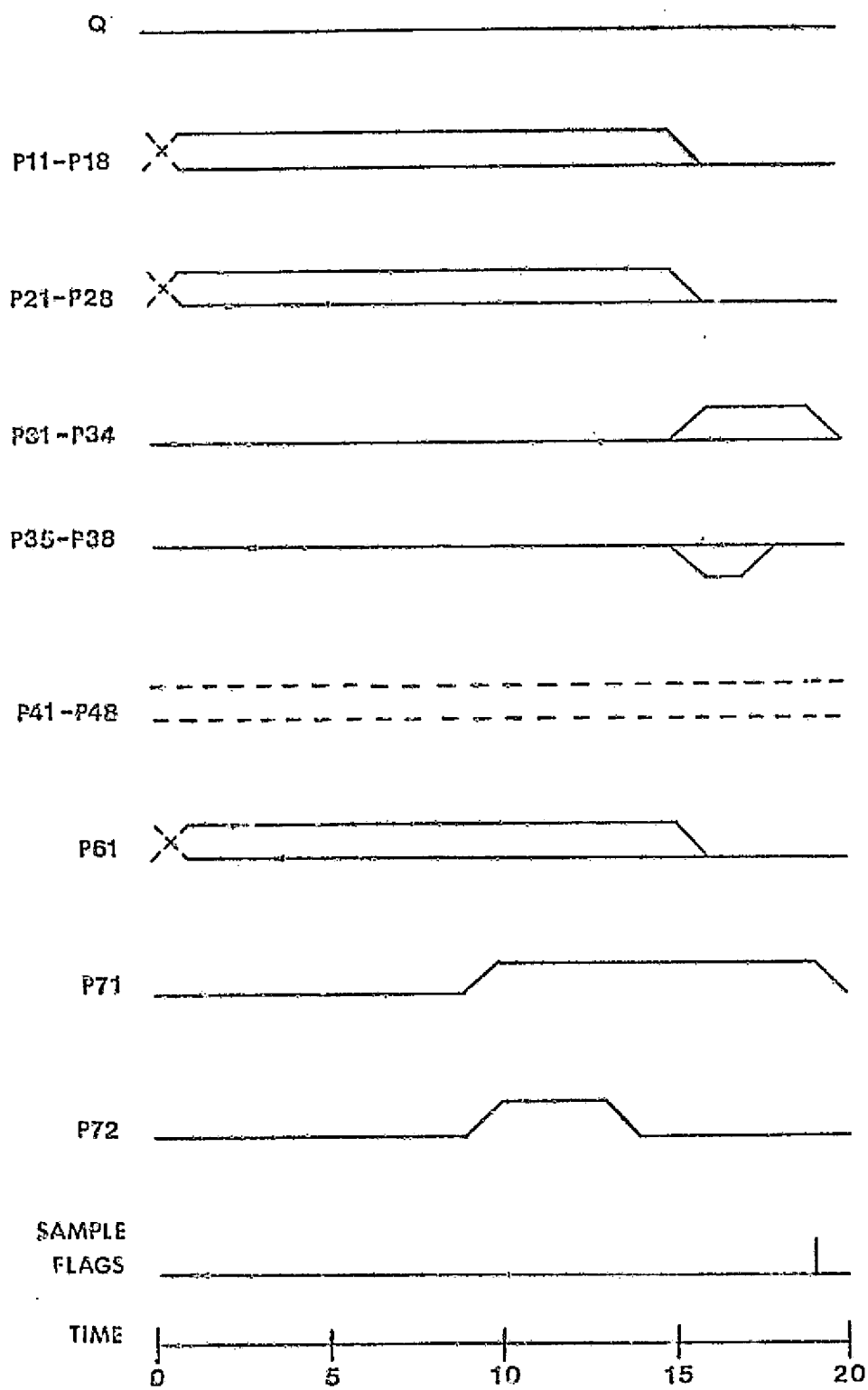


Figure 7.7 A timing diagram for the tse register and ALU operations (except TCLRI).

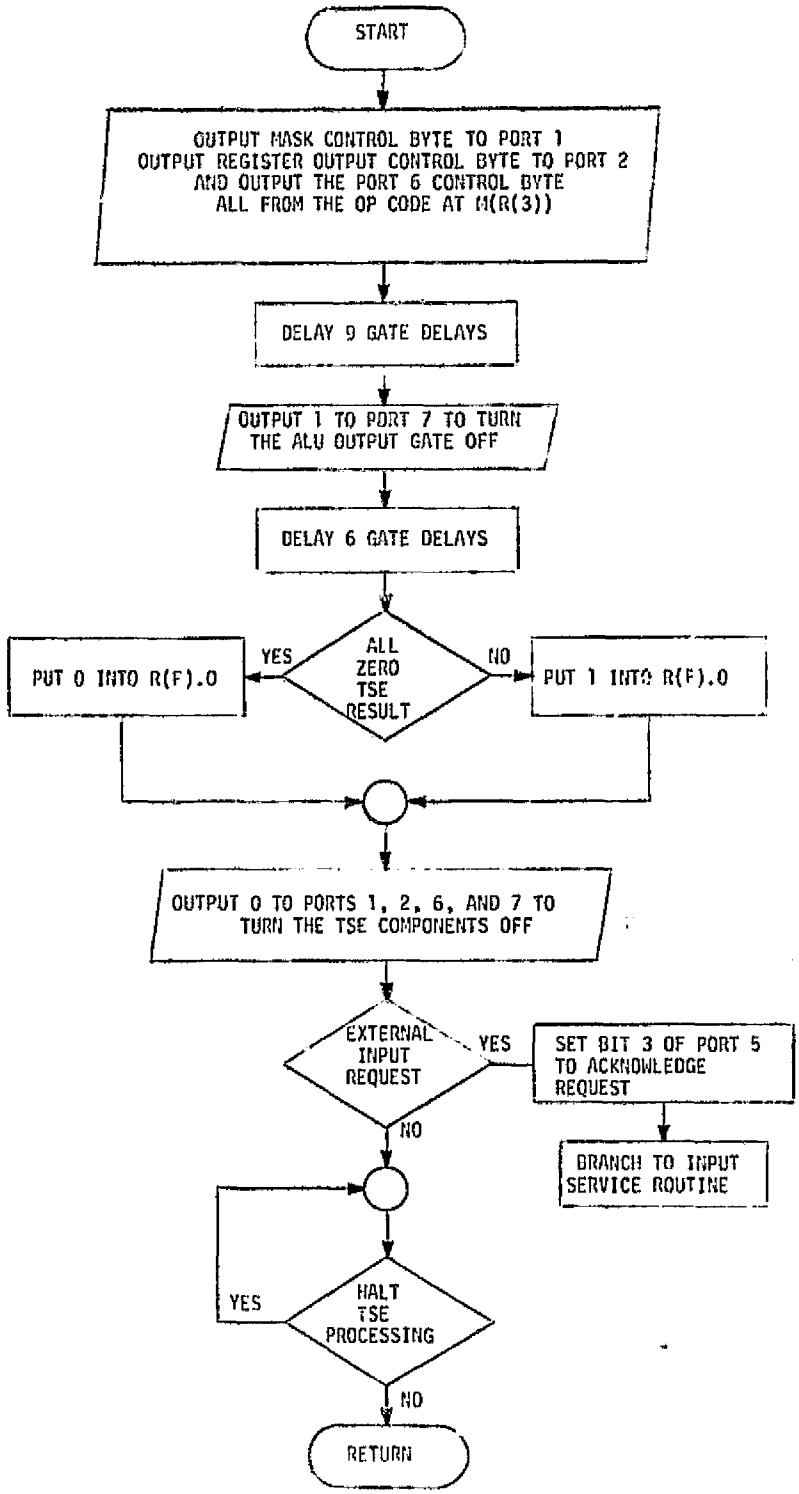


Figure 7.8 A flow chart for the tse compare operations control program.

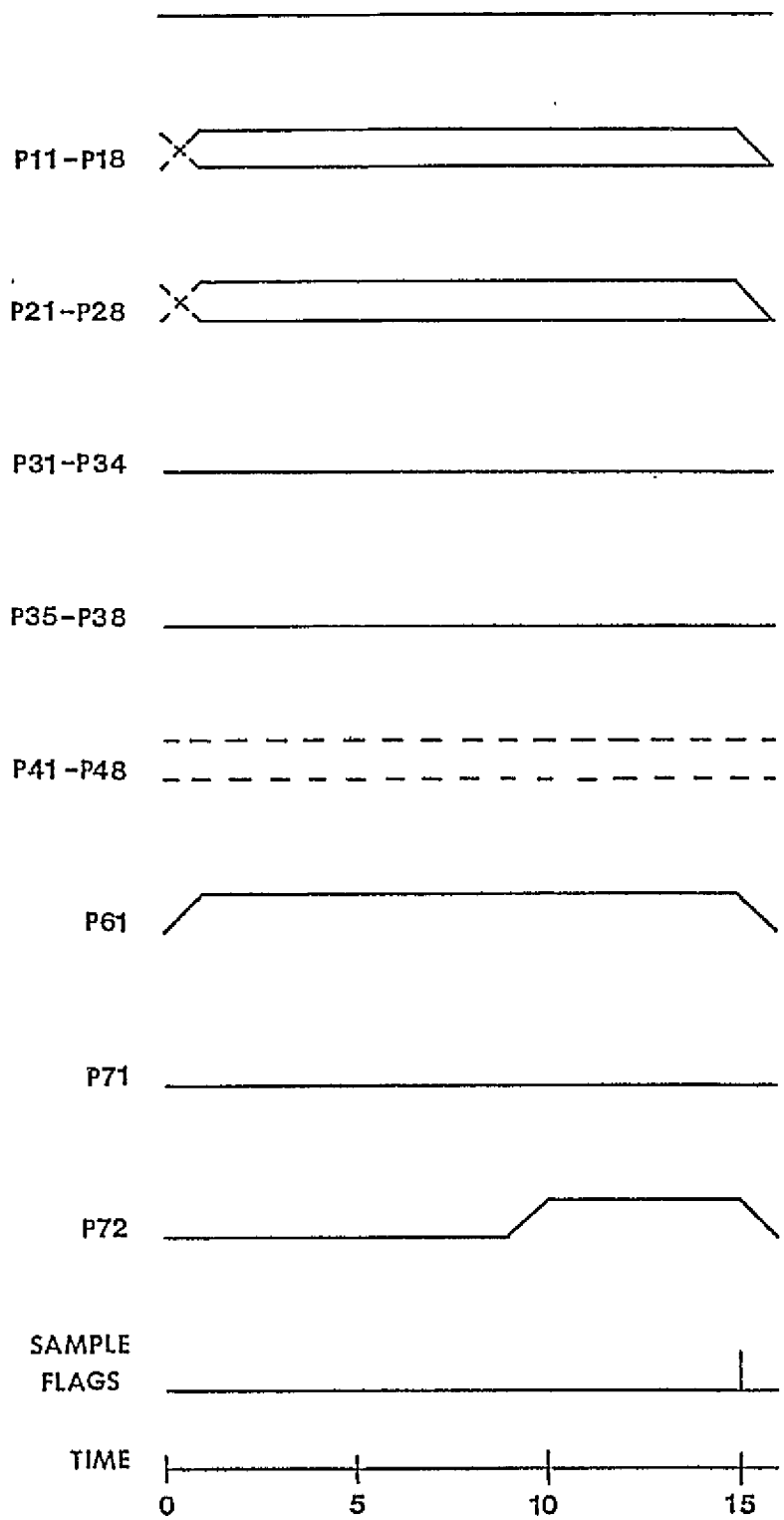


Figure 7.9 A timing diagram for the tse compare operations.

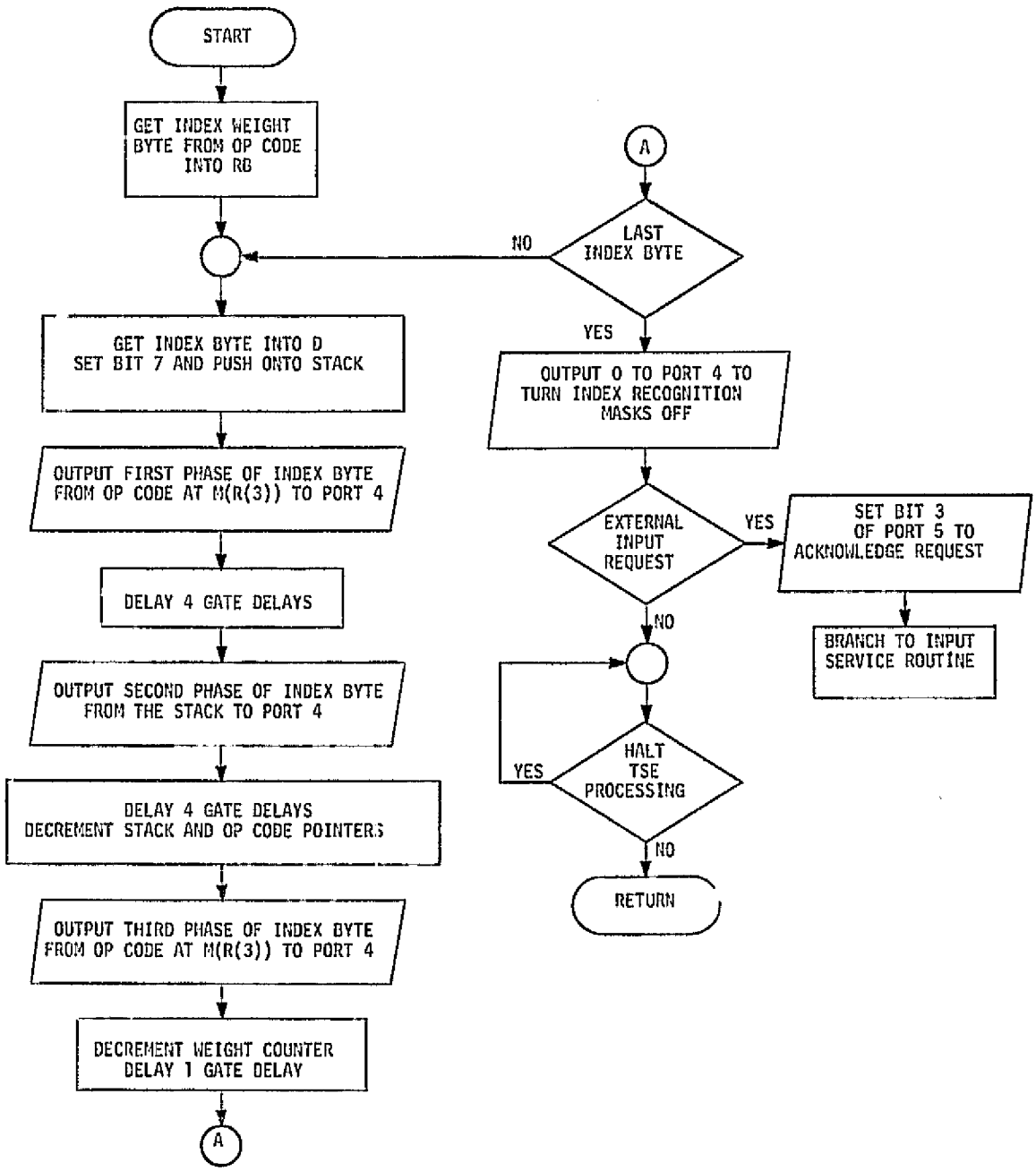


Figure 7.10 A flow chart for the index recognition control program.

operational code varies between three and eight bytes in length depending upon the weight of the index. Each index recognition instruction begins with a SEP RA. The second byte of the operational code specifies the weight of the index, and the remaining bytes provide the control bits which are output to drive the index recognition masks and enable register I. The six least significant bits of the control bytes correspond to the six Golay neighbors of a basis point but have the complementary logic state.

Index recognition images are ORed into register I so that multiple indices can be easily recognized. The index recognition instruction has a variable execution time consisting of nine gate delays for each orientation of the specified surround. A timing diagram for recognizing an index with a weight of two is shown in Figure 7.11. Most indices have a weight of six and can be recognized in 54 gate delays.

A special TCLRI instruction is provided for clearing the contents of the index recognition register before each set of index recognition operations. TCLRI is a self-contained instruction which turns the index recognition latch off by executing an 1802 OUT 4 instruction with an immediate data byte of zero. Two unit gate delays are inserted at the end of the TCLRI instruction by calling LDLY (Figure D.5, Appendix D). This insures that the index recognition latch will clear before the next index recognition operation. The TCLRI operational code is five bytes long.

The tse input instruction has a one byte operational code, SEP RD. Figure D.6, Appendix D, is a listing of the microprogram which controls the execution of the tse input instruction and Figure 7.12 is a flow

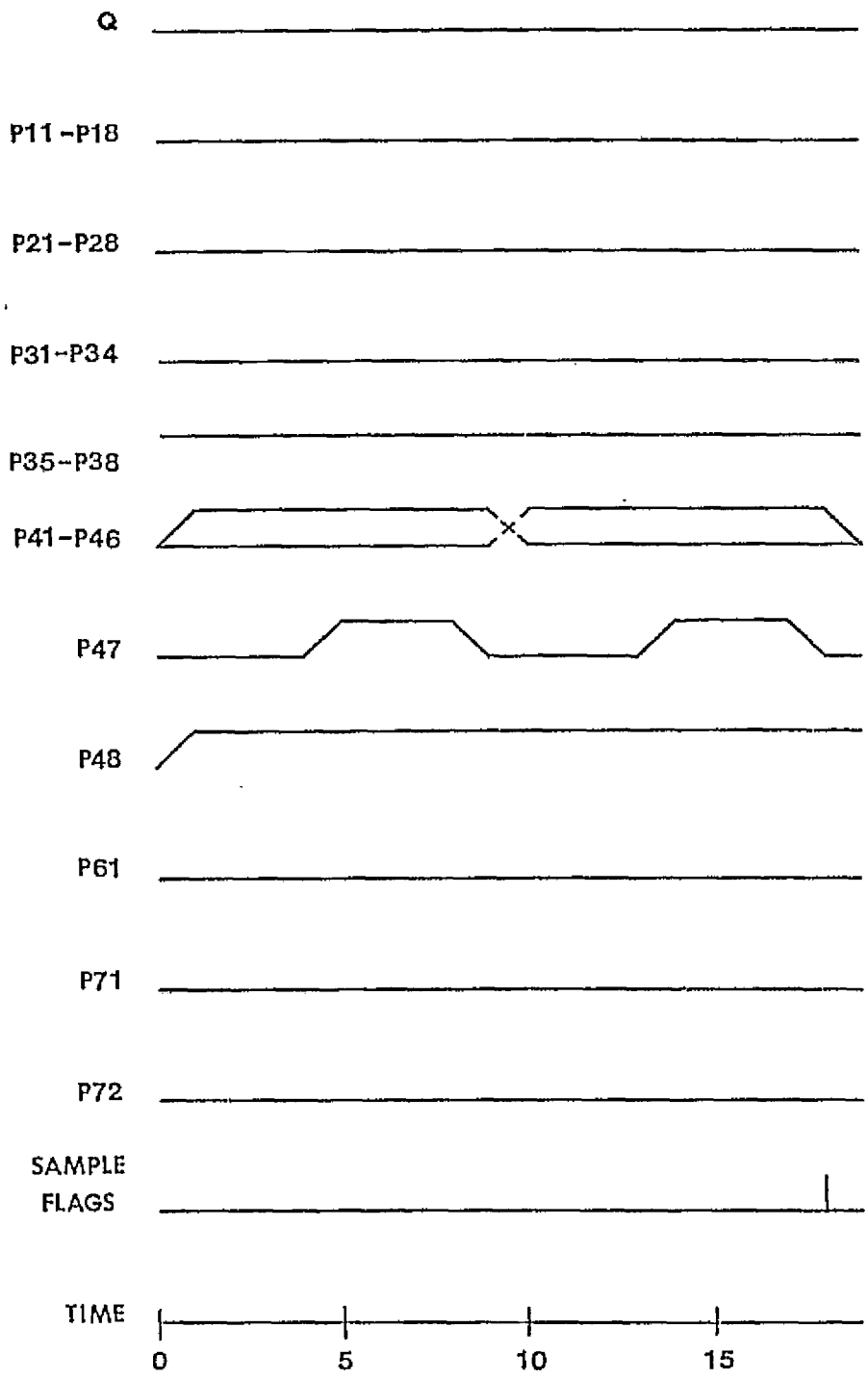


Figure 7.11 A timing diagram for recognizing an index with a weight of two.

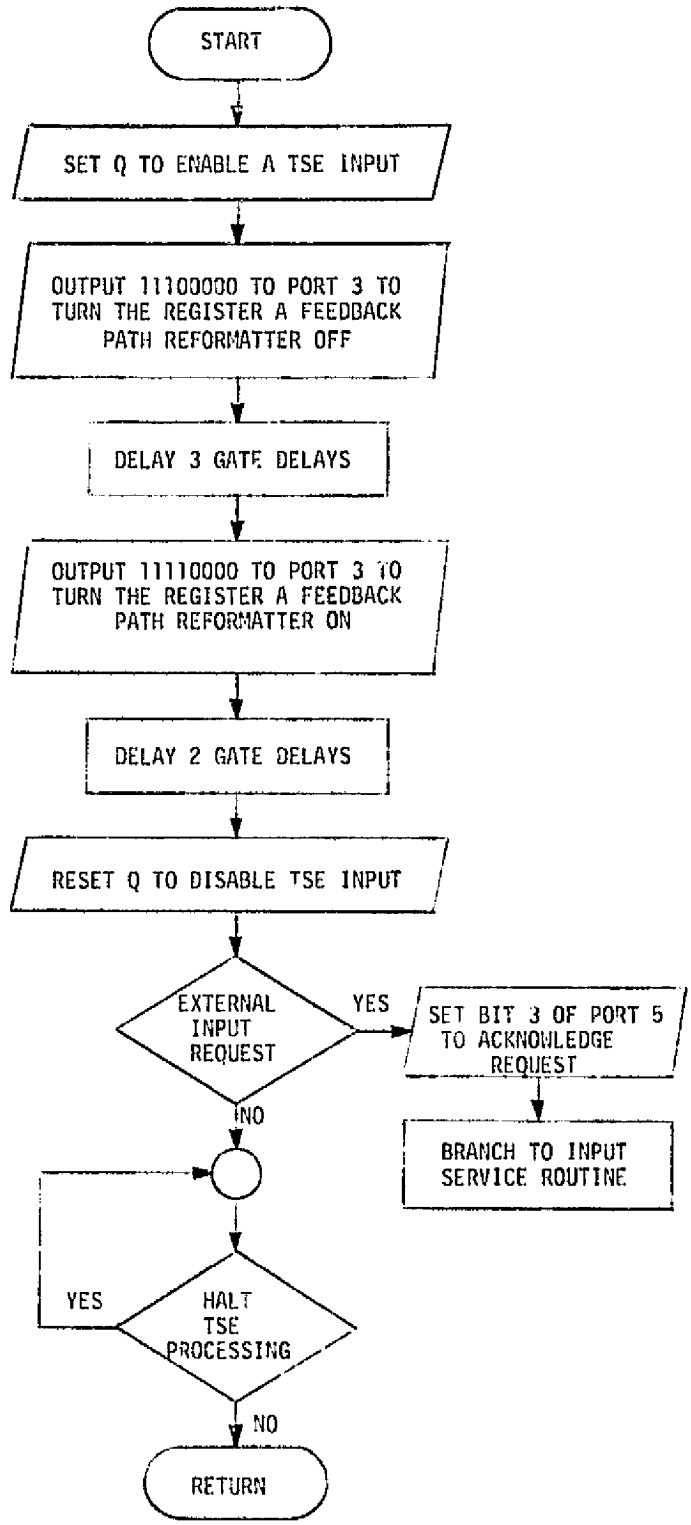


Figure 7.12 A flow chart for the tse input control program.

chart for the microprogram. The Q line is set to enable an external tse input to latch A, and the feedback path reformatter is turned off to clear the register (Figure 7.13). After three gate delays, the feedback path is re-enabled, and following two additional gate delays, Q is reset to disable the external tse input. Execution time for the tse input instruction is five gate delays. Both the EF2 and the EF? flags are tested. The tse output instruction, TOUT, is a special case of the TMOV instruction and is controlled by the general ALU operations microprogram.

Tse branch instructions are self-contained operations which perform an 1802 GLO RF instruction followed by the appropriate 1802 branch instruction (BZ, BNZ, LBZ, or LBNZ). The short tse branches require a three byte operational code, and the long tse branches require a four byte operational code. Tse branch instructions execute in essentially zero gate delays.

Table 7.4 summarizes the important characteristics of the basic tse instruction set which has been developed for the Golay transform tse computer. The efficiency of the 1802 microprogram control technique is indicated by the fact that the control routines listed in Appendix D require only 238 bytes of memory.

A Cross-Assembler for the Tse Computer

A cross-assembler has been written to aid in the development of tse computer applications programs. The tse computer cross-assembler is a macro library which can be used in conjunction with Digital Equipment Corporation's RT-11 MACRO assembler [20] and a PDP 11/40 minicom-

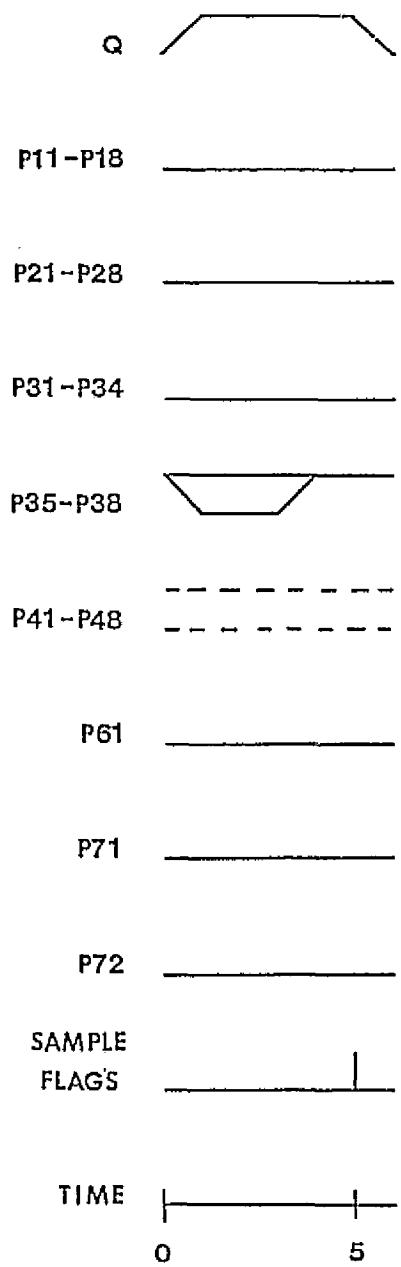


Figure 7.13 A timing diagram for the tse input operation.

TABLE 7.4

TSE INSTRUCTION CHARACTERISTICS

Instruction Class	Op Code Length in Bytes	Execution Time in Gate Delays	Control Microprogram Program Counter
Register (except TCLRI)	6	19	R9
TCLRI	5	2	-
Logic	6	19	R9
Index Recognition	3-8	9-54	RA
Compare	4	15	RC
Branch	3-4	0	-
Input	1	5	RD
Output	6	19	R9

puter to assemble both microprograms and applications programs for the tse computer. The RT-11 MACRO assembler features conditional and macroassembly capabilities [20] which are utilized in assembling programs for the tse computer. A brief summary of the RT-11 MACRO assembler commands is provided in Appendix E. Interested readers can refer to Korn [21] for a general discussion of macros and conditional assembly.

The tse computer macro library defines all of the COSMAC and tse instructions as well as the pseudo and delay instructions listed in Table 7.5. A complete listing of the tse computer macro library is too long to be included here. However, a representative subset of the macro library is listed in Appendix F. Each macro definition specifies the symbolic name of an instruction. Operational code bytes for the instruction can be calculated by the macro assembler if they cannot be specified as constants. Both logical operations and conditional tests are utilized in computing the calculated operational codes. Since the PDP 11/40 utilizes 16 bit words, the BYTE operation is used to truncate the words to eight bits. Many of the macro definitions include a call to another macro which will test for illegal conditions. For example, the short branch instruction macros call $$$PAG, which checks for an illegal branch across page boundaries.

New instructions can be added to the repertoire of the tse computer by defining additional macros for them. As an example, consider a tse mix operation that combines subfields from the accumulator and another source register to form the resultant image. This operation is given the symbolic name TMIX and assigned three arguments. The first argument

TABLE 7.5

SPECIAL TSE COMPUTER INSTRUCTIONS

Instruction	Mnemonic	Operation
PSEUDO OPERATIONS		
ORIGIN	ORG	Specifies program starting address
END	END	Marks the end of a source program
DATA BYTE	DB	Places a data byte in the object file
DATA WORD	DW	Places a data word (two bytes) in the object file
DATA STORAGE	DS	Reserves a set of memory locations for data storage
MACRO OPERATIONS		
SUBROUTINE CALL	CALL	Sets P to 4 to initiate the standard subroutine call procedure
RETURN FROM SUBROUTINE	RSR	Sets P to 5 to initiate the standard subroutine return procedure
LOAD IMMEDIATE, REGISTER	LOAD	Loads the given 16 bit constant into the specified COSMAC register
LONG DELAY, RETURN TO R (3)	LDLY	Delay 8N+22 cycles
LONG DELAY, RETURN TO CALLING PROGRAM COUNTER	MLDLY	Delay 8N+30 cycles
DELAY 3	DLY3	Delay 3 cycles
DELAY 4	DLY4	Delay 4 cycles
DELAY 6	DLY6	Delay 6 cycles
DELAY 9	DLY9	Delay 9 cycles
DELAY 10	DLY10	Delay 10 cycles
DELAY 12	DLY12	Delay 12 cycles
DELAY 15	DLY15	Delay 15 cycles
DELAY 18	DLY18	Delay 18 cycles

is REG1 which specifies the destination register. The second argument, REG2, identifies the source register that is to be mixed with the accumulator. A third argument, MASK, is required to specify the subfields of REG2 that should appear in the resultant image. The switching expression for this operation is

$$[(MASK)\cdot(REG2)]+[(\overline{MASK})\cdot(A)]\rightarrow REG1. \qquad (7)$$

Figure 7.14 provides a listing of the macro definition for the TMIX instruction.

The TMIX instruction is written to execute with R(3) as the program counter. Mask, register output, and port six control bytes are output as immediate data to gate the specified subfields for REG2 into the ALU output latch. A long delay subroutine, LDLY, (Figure D.5, Appendix D) is then called to insert a delay which allows the image to propagate through the ALU. After nine gate delays the ALU output image will consist of the specified subfields of REG2 and zeros in the unspecified subfields. This image is temporarily stored in the ALU output latch, instead of being sent to the destination register.

To complete the TMIX operation, the contents of register A should be ANDed with the complement of the subfield mask specified in the TMIX macro call and ORed with the current contents of the ALU output latch. Then the resultant image should be stored in the destination register, REG1. These operations can be executed under control of the general ALU operations microprogram (Figure D.1, Appendix D). A SEP R9 instruction byte is included in the TMIX instruction operational code to initiate the microprogram call. Conditional assembly

```
.MACRO TMIX REG1,REG2,MASK          ; MIX REGISTER WITH A
.NLIST SRC
.BYTE ^O141                         ; OUT 1
.BYTE MASK                          ; MASK CONTROL BYTE
.BYTE ^O142                         ; OUT 2
.BYTE <^O017&REG2>                  ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^O146                         ; OUT 6
.BYTE ^B00000001                    ; PORT SIX CONTROL BYTE.   P61 ON
.BYTE ^O327                         ; LDLY 2247.
DW    4307                          ; DELAY NINE GATE DELAYS
.BYTE ^O147                         ; OUT 7
.BYTE ^B00000011                    ; TURN ALU OUTPUT LATCH ON
.BYTE ^O327                         ; LDLY 997.
DW    1745                          ; DELAY FOUR GATE DELAYS
.BYTE ^O147                         ; OUT 7
.BYTE ^B00000001                    ; TURN AND GATE AT OUTPUT OF ALU OFF
.BYTE ^O327                         ; LDLY 247.
DW    0367                          ; DELAY ONE GATE DELAY
.BYTE ^O331                         ; SEP R9
.IIF EQ, MASK, BYTE ^O300           ; CONDITIONAL MASK
.IIF EQ, MASK-^O14, BYTE ^O000      ; CONTROL BYTES
.IIF EQ, MASK-^O011, BYTE ^O320
.IIF EQ, MASK-^O012, BYTE ^O340
.IIF EQ, MASK-^O017, BYTE ^O260
.IIF EQ, MASK-^O013, BYTE ^O360
.IIF EQ, MASK-^O016, BYTE ^O240
.IIF EQ, MASK-^O015, BYTE ^O220
.BYTE ^B10000001                    ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                    ; PORT SIX CONTROL BYTE.   P61 ON
.BYTE REG1                          ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^O360!REG1>                  ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TMIX
```

Figure 7.14 A macro definition for the TMIX instruction.

statements are used to establish the correct mask byte. The register output, port six, and register input bytes required by the general ALU operations microprogram are also included in the TMIX instruction operational code.

A timing diagram which illustrates the control signals for the TMIX instruction is shown in Figure 7.15. Thirty-four gate delays are required to execute the TMIX instruction. This operation could be performed by a sequence of basic tse instructions. For example,

```
TANI  C,B,M1
TORA  C,M23
```

is equivalent to

```
TMIX C,B,M1 .
```

The advantage of using the microprogrammed TMIX instruction is a reduction in execution time.

Application Program Examples

The tse computer can perform a variety of Golay transforms. Figures G.1 and G.2, Appendix G, are listings of tse computer programs for performing the Golay transform skeletonizing and swelling algorithms, respectively. The skeletonizing program is 129 bytes long and requires 607 unit gate delays to process a simple image. The swelling program is slightly more complex but requires only 147 bytes of program storage. One iteration of the swelling algorithm can be performed in 664 unit gate delays. Neither of these programs utilize tse register B. Therefore, register B can be used for temporary storage of another image. The simplicity of these programs is indicative of the power of the tse computer instruction set and CPU organization.

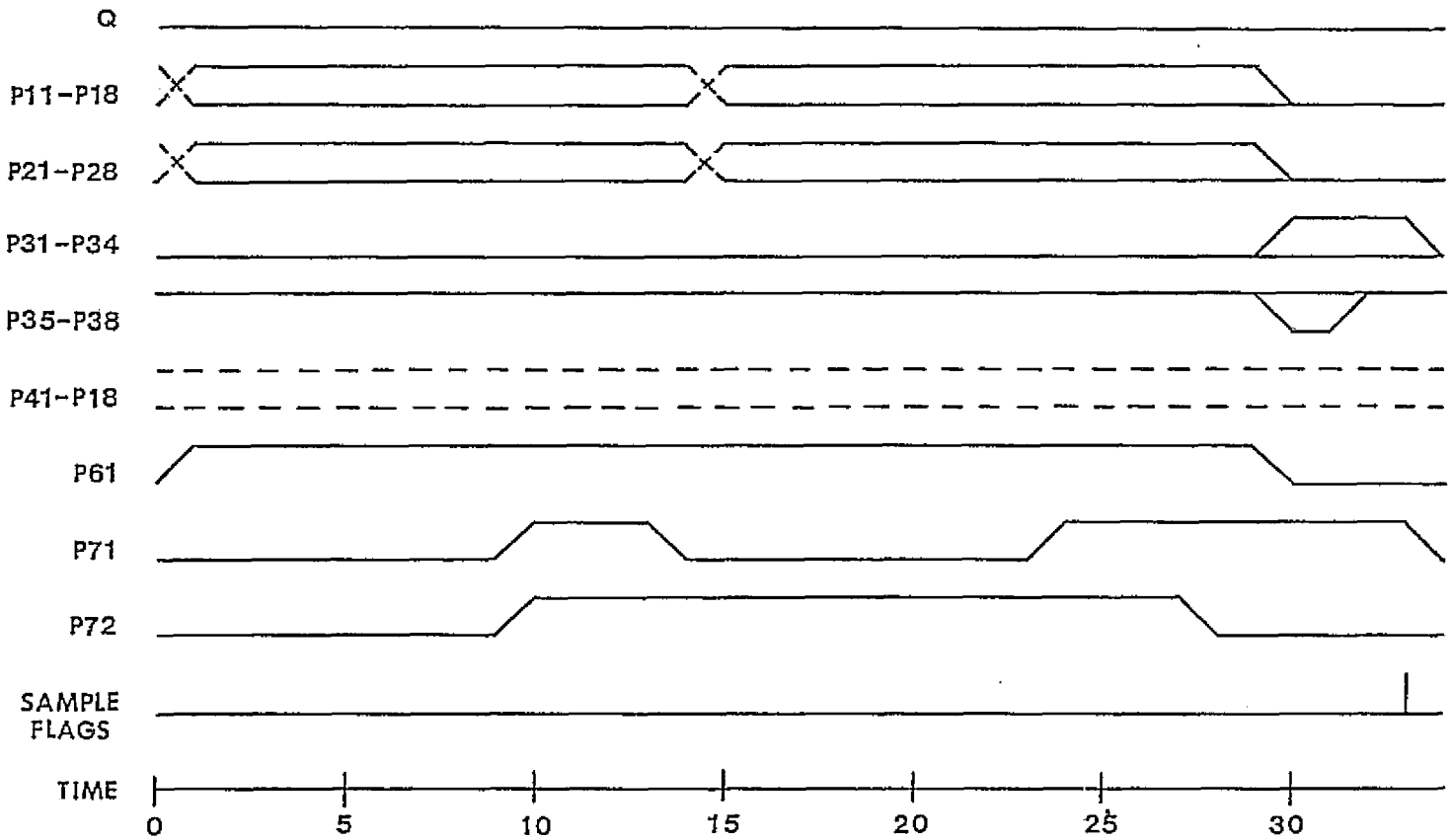


Figure 7.15 A timing diagram for the TMIX instruction.

Tse Computer Performance Evaluation

Classically, the evaluation of computer performance has proven to be a difficult problem which is highly dependent upon the task that is assigned to the computer. At this time, the performance of the special purpose Golay transform tse computer can best be evaluated by considering the skeletonizing algorithm. Table 7.6 summarizes the performance characteristics of the tse computer in the skeletonizing machine application.

The low hardware cost and relatively low data rate of this machine are primarily due to the use of a comparison type index recognition circuit. The ability of the tse computer to perform a variety of Golay transforms under program control is the primary advantage of the machine.

TABLE 7.6

PERFORMANCE OF THE TSE COMPUTER AS A SKELETONIZING MACHINE

Cost Function	Number of Control Signals	Total Component Count	Gate Delays per Iteration (Time in Seconds)	Data Rate Simple Images per Minute	Average Power Consumption in Watts	Peak Power Consumption in Watts	Speed-Power Product in Watt-Seconds
97A+69B	35	166	607 (3.04)	19.77	204.96	222	622.07

CHAPTER 8

CONCLUSION

The tse logic devices proposed by Schaefer and Strong [1] have been used to develop hardwired, pipelined, and programmable architectures for Golay transform processing machines. These machines illustrate that tse logic circuits can perform useful image processing algorithms which have not been optimized for tse logic processing. The key step toward performing Golay transforms with tse logic was the development of the Golay neighbor planes generator circuit. This circuit facilitates tse logic implementations of the index recognition operation. Because the hardware cost and processing rate of a Golay transform machine are highly dependent upon the index recognition circuit, several alternate realizations of the index recognition operation were developed. In addition, several new tse logic devices were proposed, and a set of performance evaluation parameters was developed to aid in the design of tse logic circuits. Techniques were also developed for controlling tse logic circuits with conventional logic control units constructed from a microcomputer or from programmable logic arrays.

A Critique of Tse Logic

The major advantages of tse logic devices are their ability to operate on a large number of data points simultaneously and the high data processing rates which can potentially be achieved due to this parallelism. The fan-in and fan-out limitations of the devices are

drawbacks since they tend to increase both the hardware cost and propagation delay of a tse logic circuit. Perhaps the most serious disadvantage of current tse logic circuits is the large number of device-to-device interconnections which are required. In conventional integrated logic circuits, manufacturing costs and failure rates both increase substantially with any increase in the number of external connections required by the circuit. Although improved interconnection techniques will have to be developed for tse logic circuits, there is currently no evidence to support a theory that the cost and reliability of tse logic circuits will not be heavily dependent upon the device-to-device interfaces. In fact, since the same basic integrated circuit technology is used to fabricate both conventional logic and active tse logic devices, the cost and failure rate characteristics of the conventional devices can be expected to prevail.

Suggested Directions for Future Research

One method of reducing the number of device-to-device interfaces in a tse logic circuit is to increase the functional complexity of the individual tse logic devices. This technique has been used successfully in conventional logic and should be investigated for possible use in tse logic. A projection of the integrated circuit complexity, which should be realizable by a target date such as 1985, would aid tse device designers in the task of partitioning complex logic functions into individual circuits. As an example of the usefulness of this approach, consider the potential advantage of an integrated tse latch or read-write memory. Three active and two passive tse devices are currently

required to construct the simplest tse latch. An integrated tse latch would reduce the number of device-to-device interfaces from eight to two. In addition, the integrated tse latch could potentially reduce the propagation delay, power consumption, and size of the tse latch.

Although the electro-optical family of tse logic devices was used for all the design examples in this dissertation, the basic designs and design principles described here are not dependent upon the signal transfer technique. Because of the low efficiency which is characteristic of electro-optical interfaces, additional signal transfer techniques should be investigated.

An alternate technique for achieving high speed parallel processing by using arrays of microprocessors or programmable logic circuits should also be investigated. The possible advantages of this technique include a reduction in the number of different integrated circuits which must be fabricated and reduced device-to-device interface complexity. The possibility of a reduction in interface complexity is projected because of the bus oriented structure of microprocessors.

As a long term project, research should be conducted on the use of two-dimensional tse logic control units for tse circuits. If the full potential of a completely two-dimensional computer can be realized, a revolutionary advancement over the large scale computing capabilities of today's computers could be expected.

REFERENCES

[1] D. H. Schaefer and J. P. Strong, Tse Computers, X-943-75-14, Goddard Space Flight Center, 1975.

[2] S. H. Unger, "A Computer Oriented Toward Spatial Problems," Proceedings of IRE, vol. 46, pp. 1744-1750, Oct. 1958.

[3] S. H. Unger, "Pattern Detection and Recognition," Proceedings of IRE, vol. 47, pp. 1737-1751, Oct. 1959.

[4] D. L. Slotnick, W. C. Borck, and R. C. McReynolds, "The Solomon Computer," A.F.I.P.S. Proceedings, First Fall Computer Conference, pp. 97-107, 1962.

[5] D. Lewin, Theory and Design of Digital Computers, London: Thomas Nelson and Sons LTD., 1972, pp. 308-317.

[6] J. C. Murtha, Advances in Computers, vol. 7, New York: Academic Press, 1966, pp. 10-22.

[7] "Solomon II-Parallel Network Processor," Westinghouse Electric Corp. Report No. 1869A, Aerospace Division, Baltimore, Maryland, 1964.

[8] G. H. Barnes, et. al., "The Illiac IV Computer," IEEE Trans. on Computers, vol. C-17, pp. 746-757, Aug. 1968.

[9] D. J. Kuck, "Illiac IV Software and Application Programming," IEEE Trans. on Computers, vol. C-17, pp. 758-770, Aug. 1968.

[10] M. J. E. Golay, "Hexagonal Parallel Pattern Transformations," IEEE Trans. on Computers, vol. C-18, pp. 733-739, Aug. 1969.

[11] K. Preston, "Feature Extraction by Golay Hexagonal Pattern Transforms," IEEE Trans. on Computers, vol. C-20, pp. 1007-1014, Sept. 1971.

[12] B. Kruse, "A Parallel Picture Processing Machine," IEEE Trans. on Computers, vol. C-22, pp. 1075-1086, Dec. 1973.

[13] C. D. Stamopoulos, "Parallel Image Processing," IEEE Trans. on Computers, vol. C-24, pp. 424-433, April 1975.

[14] W. H. Ware, "The Ultimate Computer," IEEE Spectrum, vol. 9, pp. 84-87, March 1972.

[15] Z. Kohavi, Switching and Finite Automata Theory, New York: McGraw-Hill, 1970, p.57.

[16] "Laser Used as Micro-Welder," Electro-Optical Systems Design, vol. 7, p.6, Jan. 1975.

[17] D. H. Schaefer, Personal communication.

[18] J. B. Peatman, The Design of Digital Systems, New York: McGraw-Hill, 1970, pp. 216-221.

[19] User Manual for the CDP1802 COSMAC Microprocessor, RCA Solid State Division, Somerville, N.J., 1976.

[20] RT-11 System Reference Manual, Digital Equipment Corporation, Maynard, Mass., 1975.

[21] G. A. Korn, Minicomputers for Engineers and Scientists, New York: McGraw-Hill, 1973, pp. 123-130.

APPENDIXES

APPENDIX A

SCHEMATIC SYMBOLS FOR TSE LOGIC DEVICES

Table A.1 lists the tse logic devices and their schematic symbols. To simplify schematic drawings, three different symbols are used for the interleaver. Note that the input and output surfaces are reversed when the interleaver is used as a combiner rather than as a duplicator. Also, note that the slide gates move an image more than one matrix position, a number should be included within the slide gate symbol to indicate the extent of the slide.

TABLE A.1

SCHEMATIC SYMBOLS FOR TSE LOGIC DEVICES

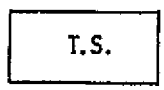
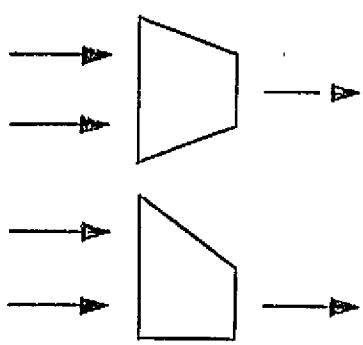
Device	Symbol
ACTIVE DEVICES	
AND	
OR	
NEGATE	
EXCLUSIVE-OR	
REFORMAT	
TOTAL SPILLER	
CONTRACTOR	
ROM	
PASSIVE DEVICES	
INTERLEAVER AS A COMBINER	

TABLE A.1
(Continued)

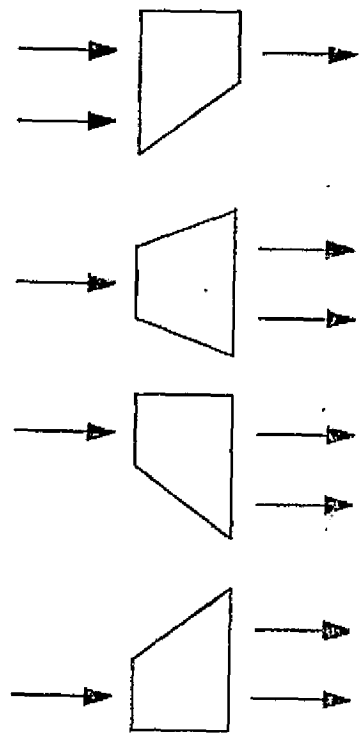
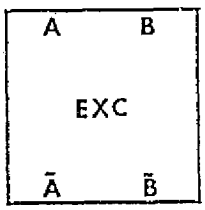
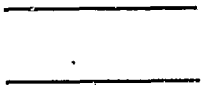
Device	Symbol
INTERLEAVER AS A DUPLICATOR	
SLIDE UP	SU
SLIDE DOWN	SD
SLIDE RIGHT	SR
SLIDE LEFT	SL

TABLE A.1
(Continued)

Device	Symbol
EXCHANGE	
IMAGE BUS-LONG	
IMAGE BUS-SHORT	

APPENDIX B

TSE MASK PATTERNS

Table B.1 defines the mask patterns for the tse read-only-memories and programmed output active devices. The ROMs are programmed to produce logic one outputs at the array positions specified as active and logic zero outputs elsewhere. In the case of programmed output active devices, the array outputs which are specified as active are normal outputs which can produce a logic one or a logic zero state that is a function of the inputs to the active device. The remaining array outputs of the programmed output active devices are disabled so that they always produce a logic zero output.

The notation G_S^T is used to specify points in the S Golay subfields of the array where the array is divided into three, four, or seven subfields as specified by the type superscript, T. For example, $G_{1,2}^3$ is the set of all points in the second and third subfields of an image where the image has been partitioned into three Golay subfields.

TABLE B.1

TSE MASK PATTERNS

Name	Symbol	Active Elements of an nxn Array with General Element a_{ij}
ALL LOGIC ONE TSE	M	All
ODD TSE MASK	MO	a_{ij} where i=0,2,4,6...
EVEN TSE MASK	ME	a_{ij} where i=1,3,5,7...
ALL LOGIC ZERO TSE	MO	None
GOLAY SUBFIELD ONE MASK	M1	$a_{ij} \varepsilon G_1^3$
GOLAY SUBFIELD TWO MASK	M2	$a_{ij} \varepsilon G_2^3$
GOLAY SUBFIELD THREE MASK	M3	$a_{ij} \varepsilon G_3^3$
GOLAY SUBFIELDS ONE AND TWO MASK	M12	$a_{ij} \varepsilon G_{1,2}^3$
GOLAY SUBFIELDS ONE AND THREE MASK	M13	$a_{ij} \varepsilon G_{1,3}^3$
GOLAY SUBFIELDS TWO AND THREE MASK	M23	$a_{ij} \varepsilon G_{2,3}^3$

APPENDIX C

THE CDP1802 MICROPROCESSOR

The CDP1802 COSMAC microprocessor [19] is a byte-oriented central processing unit constructed as a complementary-symmetry MOS integrated circuit. A block diagram of the internal structure of the 1802 (Figure C.1) shows that the COSMAC architecture is based on an array of 16 general-purpose 16-bit scratch-pad registers. These general-purpose registers are connected to a common bus and can be selected by the four-bit N, P, and X registers to perform specific tasks. The scratch-pad registers can be used as program counters, data address pointers, general-purpose counters, and temporary data storage locations. High and low bytes of the scratch-pad registers can also be gated between the register array and the eight-bit D register which functions as an accumulator.

One of the outstanding features of the COSMAC architecture is that any of the scratch-pad registers can be used as the program counter. This permits a very fast and efficient subroutine call which is performed by a one-byte set P instruction that simply specifies a new program counter. The tse computer control subroutines are called by set P instructions.

Another important feature of the 1802 is a flexible input-output structure. Four EF flags which can be used as one bit inputs are included in the CPU. These flags can be tested by 1802 branch instructions and are used to check the status of the tse computer. The Q flag functions as a single bit output which can be set, reset, and tested by

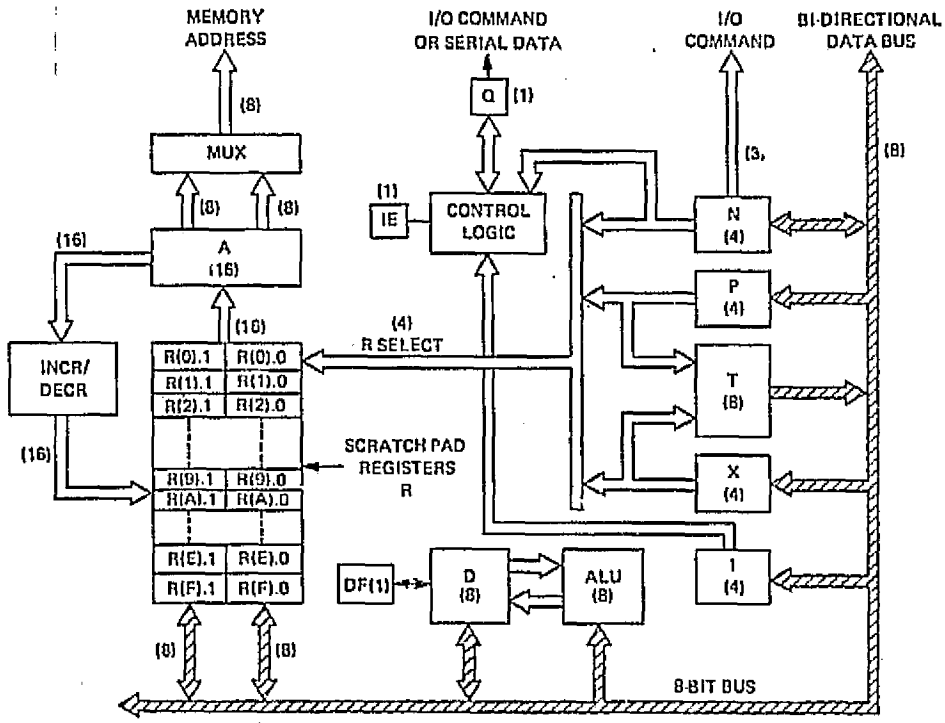


Figure C.1 Internal structure of the CDP1802 COSMAC microprocessor.

(Courtesy of Solid State Division, RCA Corporation)

the 1802 CPU. The tse computer input data path is controlled by the Q line. In addition to this on-chip I/O, the 1802 includes a set of memory-oriented I/O instructions which are used to provide control signals to the tse computer.

A summary of the COSMAC 1802 microprocessor instruction set is given in Table C.1. The notation R(W) indicates the register designated by W where W is N, X, or P. When the low order or high order bytes of R(W) are referenced individually, the notation R(W).0 refers to the low order byte while R(W).1 denotes the high order byte. As an example of the operation notation, the symbols

$$D \rightarrow M(R(X)); \; R(X)-1$$

mean that the contents of register D are stored in the memory location pointed to by the register selected by the current contents of X and that the register specified by X is decremented by one.

Several features of the 1802 instruction set should be noted. First, all COSMAC instructions except the long branch, long skip, and NOP instructions execute in two machine cycles consisting of eight states each. The long branch, long skip, and NOP instructions execute in three machine cycles. This feature simplifies the realization of precisely timed control signals. Second, most of the 1802 instructions require only a simple one-byte operational code which conserves memory. Third, all the logic and arithmetic operations utilize the contents of D and the contents of memory as operands. Data stored in scratch-pad registers cannot be operated on by these instructions. Finally, note that the memory address required by the I/O instructions is obtained from

TABLE C.1

CDP1802 MICROPROCESSOR INSTRUCTION SET [19]

Instruction	Mnemonic	Operation
MEMORY REFERENCE		
LOAD VIA N	LDN	M(R(N))→D; FOR N NOT 0
LOAD ADVANCE	LDA	M(R(N))→D; R(N)+1
LOAD VIA X	LDX	M(R(X))→D
LOAD VIA X AND ADVANCE	LDXA	M(R(X))→D; R(X)+1
LOAD IMMEDIATE	LDI	M(R(P))→D; R(P)+1
STORE VIA N	STR	D→M(R(N))
STORE VIA X AND DECREMENT	STXD	D→M(R(X)); R(X)-1
REGISTER OPERATIONS		
INCREMENT REG N	INC	R(N)+1
DECREMENT REG N	DEC	R(N)-1
INCREMENT REG X	IRX	R(X)+1
GET LOW REG N	GLO	R(N).0→D
PUT LOW REG N	PLO	D→R(N).0
GET HIGH REG N	GHI	R(N).1→D
PUT HIGH REG N	PHI	D→R(N).1
LOGIC OPERATIONS		
OR	OR	M(R(X)) OR D→D
OR IMMEDIATE	ORI	M(R(P)) OR D→D; R(P)+1
EXCLUSIVE OR	XOR	M(R(X)) XOR D→D
EXCLUSIVE OR IMMEDIATE	XRI	M(R(P)) XOR D→D; R(P)+1

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
AND	AND	M(R(X)) AND D→D
AND IMMEDIATE	ANI	M(R(P)) AND D→D; R(P)+1
SHIFT RIGHT	SHR	SHIFT D RIGHT, LSB(D)→DF, 0→MSB(D)
SHIFT RIGHT WITH CARRY	SHRC	SHIFT D RIGHT, LSB(D)→DF, DF→MSB(D)
RING SHIFT RIGHT	RSHR	
SHIFT LEFT	SHL	SHIFT D LEFT, MSB(D)→DF, 0→LSB(D)
SHIFT LEFT WITH CARRY	SHLC	SHIFT D LEFT, MSB(D)→DF, DF→LSB(D)
RING SHIFT LEFT	RSHL	
CONTROL INSTRUCTIONS		
IDLE	IDL	WAIT FOR DMA OR INTERRUPT; M(R(0))→BUS
NO OPERATION	NOP	CONTINUE
SET P	SEP	N→P
SET X	SEX	N→X
SET Q	SEQ	1→Q
RESET Q	REQ	0→Q
SAVE	SAV	T→M(R(X))
PUSH X,P TO STACK	MARK	(X,P)→T; (X,P)→M(R(2)) THEN P→X; R(2)-1
RETURN	RET	M(R(X))→(X,P); R(X)+1 1→IE
DISABLE	DIS	M(R(X))→(X,P); R(X)+1 0→IE

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
BRANCH INSTRUCTIONS - SHORT BRANCH		
SHORT BRANCH	BR	M(R(P))→R(P).0
NO SHORT BRANCH (SEE SKP)	NBR	R(P)+1
SHORT BRANCH IF D=0	BZ	IF D=0, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF D NOT 0	BNZ	IF D NOT 0, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF DF=1	BDF	IF DF=1, M(R(P))→R(P).0
SHORT BRANCH IF POS OR ZERO	BPZ	ELSE R(P)+1
SHORT BRANCH IF EQUAL OR GREATER	BGE	
SHORT BRANCH IF DF=0	BNF	IF DF=0, M(R(P))→(P).0
SHORT BRANCH IF MINUS	BM	ELSE R(P)+1
SHORT BRANCH IF LESS	BL	
SHORT BRANCH IF Q=1	BQ	IF Q=1, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF Q=0	BNQ	IF Q=0, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF1=1	B1	IF EF1=1, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF1=0	BN1	IF EF1=0, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF2=1	B2	IF EF2=1, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF2=0	BN2	IF EF2=0, M(R(P))→R(P).0 ELSE R(P)+1

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
SHORT BRANCH IF EF3=1	B3	IF EF3=1, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF3=0	BN3	IF EF3=0, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF4=1	B4	IF EF4=1, M(R(P))→R(P).0 ELSE R(P)+1
SHORT BRANCH IF EF4=0	BN4	IF EF4=0, M(R(P))→R(P).0 ELSE R(P)+1
INPUT - OUTPUT BYTE TRANSFER		
OUTPUT 1	OUT 1	M(R(X))→BUS; R(X)+1; N LINES=1
OUTPUT 2	OUT 2	M(R(X))→BUS; R(X)+1; N LINES=2
OUTPUT 3	OUT 3	M(R(X))→BUS; R(X)+1; N LINES=3
OUTPUT 4	OUT 4	M(R(X))→BUS; R(X)+1; N LINES=4
OUTPUT 5	OUT 5	M(R(X))→BUS; R(X)+1; N LINES=5
OUTPUT 6	OUT 6	M(R(X))→BUS; R(X)+1; N LINES=6
OUTPUT 7	OUT 7	M(R(X))→BUS; R(X)+1; N LINES=7
INPUT 1	INP 1	BUS→M(R(X)); BUS→D; N LINES=1
INPUT 2	INP 2	BUS→M(R(X)); BUS→D; N LINES=2
INPUT 3	INP 3	BUS→M(R(X)); BUS→D; N LINES=3
INPUT 4	INP 4	BUS→M(R(X)); BUS→D; N LINES=4
INPUT 5	INP 5	BUS→M(R(X)); BUS→D; N LINES=5
INPUT 6	INP 6	BUS→M(R(X)); BUS→D; N LINES=6
INPUT 7	INP 7	BUS→M(R(X)); BUS→D; N LINES=7

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
BRANCH INSTRUCTIONS - LONG BRANCH		
LONG BRANCH	LBR	M(R(P))→R(P).1 M(R(P)+1)→R(P).0
NO LONG BRANCH (SEE LSKP)	NLBR	R(P)+2
LONG BRANCH IF D=0	LBZ	IF D=0, M(R(P))→R(P).1 M(R(P))+1)→R(P).0 ELSE R(P)+2
LONG BRANCH IF D NOT 0	LBNZ	IF DO NOT 0, M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
LONG BRANCH IF DF=1	LBDF	IF DF=1, M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
LONG BRANCH IF DF=0	LBNF	IF DF=0, M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
LONG BRANCH IF Q=1	LBQ	IF Q=1, M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
LONG BRANCH IF Q=0	LBNQ	IF Q=0, M(R(P))→R(P).1 M(R(P)+1)→R(P).0 ELSE R(P)+2
SKIP INSTRUCTIONS		
SHORT SKIP (SEE NBR)	SKP	R(P)+1
LONG SKIP (SEE NLBR)	LSKP	R(P)+2

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
LONG SKIP IF D=0	LSZ	IF D=0, R(P)+2 ELSE CONTINUE
LONG SKIP IF D NOT 0	LSNZ	IF D NOT 0, R(P)+2 ELSE CONTINUE
LONG SKIP IF DF=1	LSDF	IF DF=1, R(P)+2 ELSE CONTINUE
LONG SKIP IF DF=0	LSNF	IF DF=0, R(P)+2 ELSE CONTINUE
LONG SKIP IF Q=1	LSQ	IF Q=1, R(P)+2 ELSE CONTINUE
LONG SKIP IF Q=0	LSNQ	IF Q=0, R(P)+2 ELSE CONTINUE
LONG SKIP IF IE=1	LSIE	IF IE=1, R(P)+2 ELSE CONTINUE
ARITHMETIC OPERATIONS		
ADD	ADD	M(R(X))+D→DF, D
ADD IMMEDIATE	ADI	M(R(P))+D→DF, D; R(P)+1
ADD WITH CARRY	ADC	M(R(X))+D+DF→DF, D
ADD WITH CARRY, IMMEDIATE	ADCI	M(R(P))+D+DF→DF, D R(P)+1
SUBTRACT D	SD	M(R(X))-D→DF, D
SUBTRACT D IMMEDIATE	SDI	M(R(P))-D→DF, D; R(P)+1
SUBTRACT D WITH BORROW	SDB	M(R(X))-D-(NOT DF)→DF, D
SUBTRACT D WITH BORROW, IMMEDIATE	SDBI	M(R(P))-D-(NOT DF)→DF, D; R(P)+1
SUBTRACT MEMORY	SM	D-M(R(X))→DF, D

TABLE C.1
(Continued)

Instruction	Mnemonic	Operation
SUBTRACT MEMORY, IMMEDIATE	SMI	D-M(R(P))→DF, D; R(P)+1
SUBTRACT MEMORY WITH BORROW	SMB	D-M(R(X))-(NOT DF)→DF, D
SUBTRACT MEMORY WITH BORROW, IMMEDIATE	SMBI	D-M(R(P))-(NOT DF)→DF, D R(P)+1

R(X). Since the contents of register X can be changed by a one-byte set X instruction, data can be output efficiently from calling programs, data storage areas in memory, and from immediate data bytes. This feature is utilized extensively in the tse computers control programs which output both immediate data and data obtained from the calling program.

In some applications a subroutine is called from several distinct programs which may use different program counters. The set P subroutine call technique is unsatisfactory for these applications because the subroutine cannot easily determine which register was the calling program counter. Two alternate subroutine call procedures are provided to simplify this type of subroutine call. The first procedure is a MARK subroutine technique [19] in which the MARK instruction is used to save the current value of X and P in a software stack. This method permits the use of nested subroutines where the nesting order varies dynamically. The second procedure is the standard call and return technique [19] which uses two linking subroutines to control the call and return processes. The standard call and return technique is the most advanced call and return method. Advantages of the standard call and return technique include unlimited subroutine nesting capability and maximum flexibility in storing scratch-pad registers. In the standard subroutine call and return technique, registers four and five are assumed to point to the linking call subroutine and the linking return subroutine, respectively. A call is initiated by setting P to four. The address of the called subroutine is specified by two data bytes which should follow the set P instruction. Returns are initiated by setting P to five. Except during the actual call and return operations,

both main programs and subroutines which utilize the standard call and return technique execute with register three as the program counter. All three subroutine call procedures are used in the tse computer programs to maximize their efficiency.

The standard subroutine call and return technique requires some of the scratch-pad registers to be dedicated to particular functions. The tse computer control programs also assign particular functions to certain scratch-pad registers. Table C.2 lists the functions assigned to the COSMAC registers in the tse computer control unit application.

TABLE C.2

COSMAC CDP1802 REGISTER ASSIGNMENTS

Register	Function
R(0)	DMA Address Register
R(1)	Interrupt Service Program Counter
R(2)	Stack Pointer
R(3)	Main Program Counter
R(4)	Dedicated Program Counter for the Call Subroutine
R(5)	Dedicated Program Counter for the Return Subroutine
R(6)	Pointer to the Return Location and Arguments Passed to the Called Subroutine
R(7)	Dedicated Program Counter for the Long Delay Subroutine with R(3) as the Calling Program Counter
R(8)	Scratch-Pad Register used by the Long Delay Subroutine
R(9)	Dedicated Program Counter for the General ALU Operations Control Program
R(A)	Dedicated Program Counter for the Index Recognition Control Program
R(B)	Unassigned
R(C)	Dedicated Program Counter for the Compare Operations Control Program
R(D)	Dedicated Program Counter for the Input Control Program
R(E)	Dedicated Program Counter for the Long Delay Subroutine for Variable Calling Program Counters
R(F)	Unassigned

APPENDIX D

CONTROL MICROPROGRAMS FOR THE TSE COMPUTER

This appendix lists four microprograms which control execution of the basic tse instruction set of the Golay transform tse computer. Two long delay subroutines are also listed. The total length of the microprograms presented in this appendix is 238 bytes.

```
                ;       TSE COMPUTER GENERAL ALU OPERATIONS CONTROL PROGRAM

                ;        ALL INSTRUCTIONS EXCEPT TCLRI, TIDA, TCNT, TTEST,
                ;           TCMP, TCPI, TBZ, TBNZ, TLBZ, TLBNZ AND IIN.
                ;             R9 FUNCTIONS AS THE PROGRAM COUNTER

010000          EXALU:  SEP     R3
010000  323
010001          ALUOP:  OUT     1               ; OUTPUT MASK CONTROL BYTE
010001  141
010002                  OUT     2               ; OUTPUT REGISTER OUTPUT CONTROL BYTE
010002  142
010003                  OUT     6               ; OUTPUT PORT 6 CONTROL BYTE
010003  146
010004                  MLDLY   2247.           ; DELAY 9 GATE DELAYS
010004  171
010005  336
010006  010
010007  307
010010  042
010011          DEST:   SEX     R9              ; PREPARE TO OUTPUT IMMEDIATE DATA
010011  351
010012                  OUT     7               ; TURN AND GATE AT OUTPUT OF ALU ON
010012  147
010013                  DB      <^B00000011>    ; AND TURN ALU OUTPUT LATCH FEEDBACK
010013  003
                                                ; PATH ON
```

Figure D.1 Tse computer general ALU operations control program.

```
010014                  MLDLY   997.            ; DELAY 4 GATE DELAYS
010014  171
010015  336
010016  003
010017  345
010020  042
010021                  OUT     7               ; TURN AND GATE AT OUTPUT OF
010021  147
010022                  DB      <^B000000001>   ; ALU OFF
010022  001
010023                  MLDLY   496.            ; DELAY TWO GATE DELAYS
010023  171
010024  336
010025  001
010026  360
010027  042
010030                  OUT     1               ; TURN ALU MASK ROM'S OFF
010030  141
010031                  DB      <^B000000000>
010031  000
010032                  OUT     2               ; TURN REGISTER OUTPUTS AND REMAINING
010032  142
010033                  DB      <^B000000000>   ; ALU MASKS OFF
010033  000
010034                  OUT     6               ; TURN P61 OFF
010034  146
010035                  DB      <^B000000000>   ;
010035  000
```

Figure D.1 (Continued)

```
010036                  SEX     R3              ; PREPARE TO OUTPUT DATA FROM MAIN
010036  343                                     ; PROGRAM
010037                  OUT     3               ; TURN THE REFORMATTER AT THE INPUT
010037  143                                     ; OF THE SELECTED LATCH ON AND TURN
                                                ; THE FEEDBACK PATH REFORMATTER OF THAT
                                                ; LATCH OFF IF AN OR OPERATION IS
                                                ; NOT REQUIRED
010040                  MLDLY   497.            ; DELAY 2 GATE DELAYS
010040  171
010041  336
010042  001
010043  361
010044  042
010045                  OUT     3               ; TURN FEEDBACK PATH REFORMATTER
010045  143                                     ; BACK ON IF IT WAS TURNED OFF
010046                  MLDLY   495.            ; DELAY TWO GATE DELAYS
010046  171
010047  336
010050  001
010051  357
010052  042
010053                  B1      DST1            ; TEST CONTRACTOR OUTPUT WHICH IS
010053  064
010054  062                                     ; WIRED TO NOT EF1, BRANCH IF EF1=1
```

Figure D.1 (Continued)

```
010055                  LDI     1               ; 1-->D
010055  370
010056  001
010057                  PLO     RF              ; 1-->RF CURRENT RESULT IS NOT AN ALL
010057  257
                                                ; ZERO TSE
010060                  BR      DST2            ; BRANCH TO CONTINUE DESTINATION PROGRAM
010060  060
010061  066
010062      DST1:       LDI     0               ; 0-->D
010062  370
010063  000
010064                  PLO     RF              ; 0-->RF CURRENT RESULT IS AN ALL
010064  257
                                                ; ZERO TSE
010065                  PLO     RF              ; DELAY 2 CYCLES
010065  257
010066      DST2:       SEX     R9              ; PREPARE TO OUTPUT IMMEDIATE DATA
010066  351
010067                  DLY6                    ; DELAY 6 STATES
010067  304
010070  304
010071                  OUT     3               ; TURN THE INPUT TO EVERY LATCH OFF
010071  143
010072                  DB      <^B11110000>    ; WITH FEEDBACK PATHS ALL LEFT ON
010072  360
010073                  OUT     7               ; TURN ALU OUTPUT LATCH OFF
010073  147
010074                  DB      <^B00000000>
010074  000
```

Figure D.1 (Continued)

```
010075                  B2      DST3            ; CHECK FOR EXTERNAL INPUT REQUEST
010075  065
010076  104
010077                  OUT     5               ; IF REQUEST IS PRESENT ACKNOWLEDGE
010077  145
                                                ; BY SETTING BIT 3 OF PORT 5.
010100                  DB      <^B00000100>    ; THE INPUT ROUTINE MUST RESET THIS
010100  004
010101                  LBR     INPUT           ; BIT.
010101  300
010102  017
010103  377
010104          DST3:   B3      DST4            ; OTHERWISE, CHECK FOR A REQUEST TO HALT
010104  066
010105  110
010106          *       BR      DST3            ; TSE PROCESSING AND GO INTO PROGRAM LOOP
010106  060
010107  104
                                                ; IF THE REQUEST IS PRESENT
010110          DST4:   SEX     R3              ; RETURN VALUE OF X TO 3
010110  343
010111                  BR      EXALU           ; BRANCH TO JUST BEFORE THE ENTRY
010111  060
010112  000
                                                ; POINT TO RESTORE R9 BEFORE
                                                ; RETURNING TO THE MAIN PROGRAM
```

Figure D.1 (Continued)

```
                ;       LONG DELAY SUBROUTINE FOR VARIABLE PROGRAM COUNTERS
                ;                GENERATES 8N+30 CYCLES DELAY
                ;             RE FUNCTIONS AS THE PROGRAM COUNTER

010322          EXMLDY: RET                     ; RETURN TO CALLING PROGRAM
010322  160
010323          MLDLY:  LDXA
010323  162
010324                  PHI     R8              ; PUT HIGH AND LOW BYTES OF THE
010324  270
010325                  LDXA                    ; DELAY CONSTANT INTO R8
010325  162
010326                  PLO     R8              ;
010326  250
010327          MLDLY1: DEC     R8              ; DECREMENT R8 IN A LOOP
010327  050
010330                  GHI     R8              ; HIGH BYTE OF R8-->D
010330  230
010331                  XRI     377             ; COMPLEMENT D
010331  373
010332  377
010333                  BNZ     MLDLY1          ; BRANCH TO REPEAT IF NOT ZERO
010333  072
010334  327
010335                  SEX     2               ; ELSE SET X TO 2
010335  342
010336                  INC     R2              ; INCREMENT R2
010336  022
010337                  BR      EXMLDY          ; BRANCH TO ENTRY POINT MINUS 1
010337  060
010340  322
```

Figure D.2 Long delay subroutine for variable program counters.

```
                ;           TSE COMPUTER COMPARE OPERATIONS CONTROL PROGRAM
                ;               INSTRUCTIONS TCNT, TTEST, TCMP, AND TCPI
                ;                  RC IS USED AS THE PROGRAM COUNTER

010200          EXCMP:  SEP     R3              ; RETURN TO MAIN PROGRAM
010200  323
010201          CMPOP:  OUT     1               ; OUTPUT MASK CONTROL BYTE
010201  141
010202                  OUT     2               ; OUTPUT REGISTER OUTPUT CONTROL BYTE
010202  142
010203                  OUT     6               ; OUTPUT PORT 6 CONTROL BYTE
010203  146
010204                  MLDLY   2247.           ; DELAY 9 GATE DELAYS
010204  171
010205  336
010206  010
010207  307
010210  042
010211                  SEX     R9              ; PREPARE TO OUTPUT IMMEDIATE DATA
010211  351
010212                  OUT     7               ; TURN AND GATE AT OUTPUT OF ALU ON
010212  147
010213                  DB      <^B00000001>    ; AND LEAVE THE ALU OUTPUT LATCH FEEDBACK
010213  001
                                                ; PATH REFORMATTER OFF
010214                  MLDLY   1497.           ; DELAY 6 GATE DELAYS
010214  171
010215  336
010216  005
010217  331
010220  042
```

Figure D.3 Tse computer compare operations control program.

```
010221                      B1      CMP1            ; TEST THE CONTRACTOR OUTPUT WHICH
010221  064
010222  230
                                                    ; IS WIRED TO NOT EF1.  BRANCH IF EF1=1
010223                      LDI     1               ; 1-->D
010223  370
010224  001
010225                      PLO     RF              ; 1-->RF CURRENT RESULT IS NOT AN ALL
010225  257
                                                    ; ZERO TSE
010226                      BR      CMP2            ; BRANCH TO CHECK FOR EXTERNAL INPUT
010226  060
010227  234
                                                    ; REQUEST
010230          CMP1:       LDI     0               ; 0-->D
010230  370
010231  000
010232                      PLO     RF              ; 0-->RF
010232  257
010233                      PLO     RF              ; DELAY TWO CYCLES
010233  257
010234          CMP2:       OUT     1               ; TURN ALU MASKS OFF
010234  141
010235                      DB      <^B00000000>    ;
010235  000
010236                      OUT     2               ; TURN REGISTER OUTPUTS AND REMAINING
010236  142
010237                      DB      <^B00000000>    ; ALU MASKS OFF
010237  000
010240                      OUT     6               ; TURN PORT 6 CONTROL BITS OFF
010240  146
```

Figure D.3 (Continued)

```
010241                  DB      <^B00000000>    ;
010241  000
010242                  OUT     7               ; TURN AND GATE AT THE OUTPUT OF
010242  147
010243                  DB      <^B00000000>    ; THE ALU OFF
010243  000
010244                  B2      CMP3            ; CHECK FOR EXTERNAL INPUT REQUEST
010244  065
010245  253
010246                  OUT     5               ; IF THE REQUEST IS PRESENT ACKNOWLEDGE
010246  145
010247                  DB      <^B00000100>    ; BY SETTING BIT 3 OF PORT 5.  INPUT
010247  004
010250                  LBR     INPUT           ; ROUTINE MUST RESET THIS BIT
010250  300
010251  017
010252  377
010253      CMP3:       B3      CMP4            ; CHECK FOR A REQUEST TO HALT TSE
010253  066
010254  257
010255                  BR      CMP3            ; PROCESSING AND GO INTO A PROGRAM LOOP
010255  060
010256  253
                                                ; IF THE REQUEST IS PRESENT
010257      CMP4:       SEX     R3              ; OTHERWISE RETURN THE VALUE OF X TO 3
010257  343
                                                ; AND BRANCH TO JUST BEFORE THE ENTRY
010260                  BR      EXCMP           ; POINT TO RESTORE RC BEFORE RETURNING
010260  060
010261  200
                                                ; TO THE MAIN PROGRAM.
```

Figure D.3 (Continued)

```
                ;                   INDEX RECOGNITION CONTROL PROGRAM

                ;                                   INSTRUCTION TIDA
                ;     RA IS THE PROGRAM COUNTER.   RB IS USED AS A WEIGHT COUNTER.


010113          EXID:   SEP     R3
010113  323
010114          IDAOP:  LDXA                    ; GET INDEX WEIGHT INTO D
010114  162
010115                  PLO     RB              ; STORE WEIGHT IN REGISTER B
010115  253
010116          IDAOP1: LDN     R3              ; LOAD INDEX BYTE INTO D
010116  003
010117                  ORI     100             ; SET BIT 7 OF INDEX BYTE IN D
010117  371
010120  100
010121                  STR     R2              ; PUSH SECOND PHASE OF INDEX BYTE ONTO
010121  122
                                                ; STACK
010122                  OUT     4               ; OUTPUT INDEX IDENTIFICATION CONTROL
010122  144
                                                ; BYTE
010123                  MLDLY   996.            ; DELAY 4 GATE DELAYS
010123  171
010124  336
010125  003
010126  344
010127  042
```

Figure D.4 Tse computer index recognition control program.

```
010130                  SEX     R2              ; PREPARE TO OUTPUT DATA FROM THE STACK
010130  342
010131                  DLY6                    ; DELAY 6 CYCLES
010131  304
010132  304
010133                  OUT     4               ; OUTPUT SECOND PHASE OF INDEX
010133  144
                                                ; IDENTIFICATION
                                                ; CONTROL BYTE TO ENABLE INPUT TO
                                                ; LATCH I
010134                  DEC     R2              ; RESTORE STACK POINTER TO CORRECT
010134  042
                                                ; VALUE
010135                  DEC     R3              ; POINT R3 BACK TO CURRENT INDEX
010135  043
                                                ; RECOGNITION BYTE
010136                  MLDLY   996.            ; DELAY 4 GATE DELAYS
010136  171
010137  336
010140  003
010141  344
010142  042
010143                  SEX     R3              ; DELAY 2 CYCLES
010143  343
010144                  SEX     R3              ; PREPARE TO OUTPUT INDEX RECOGNITION
010144  343
                                                ; BYTE AGAIN
010145                  OUT     4               ; DISABLE INPUT TO LATCH I
010145  144
010146                  DEC     RB              ; DECREMENT WEIGHT COUNTER
010146  053
```

Figure D.4 (Continued)

```
010147                  MLDLY   246.            ; DELAY 1 GATE DELAY
010147  171
010150  336
010151  000
010152  366
010153  042
010154                  GLO     RB              ; GET LOW BYTE OF WEIGHT COUNTER INTO D
010154  213
010155                  BZ      IDAOP1          ; BRANCH TO CHECK NEXT ORIENTATION IF
010155  062
010156  116                                     ; NOT FINISHED
010157                  SEX     RA              ; PREPARE TO OUTPUT IMMEDIATE DATA
010157  352
010160                  OUT     4               ; TURN INDEX RECOGNITION MASKS OFF
010160  144
010161                  DB      <^B00000000>
010161  000
010162                  BZ      IDA1            ; CHECK FOR EXTERNAL INPUT REQUEST
010162  065
010163  171
010164                  OUT     5               ; ACKNOWLEDGE REQUEST BY SETTING BIT
010164  145                                     ; 3 OF PORT 5.
010165                  DB      <^B00000100>    ; THE INPUT SERVICE ROUTINE
010165  004                                     ; MUST RESET THIS BIT.
010166                  LBR     INPUT           ; BRANCH TO INPUT SERVICE ROUTINE
010166  300
010167  017
010170  377
```

Figure D.4 (Continued)

```
010171          IDA1:   B3      IDA2            ; CHECK FOR A REQUEST TO HALT TSE
010171  066
010172  175
                                                ; PROCESSING
010173                  BR      IDA1            ; GO INTO A PROGRAM LOOP IF REQUEST
010173  060
010174  171
                                                ; IS PRESENT
010175          IDA2:   SEX     R3              ; PREPARE TO RETURN TO THE MAIN PROGRAM
010175  343
                                                ; WITH X=3
010176                  BR      EXID            ; BRANCH TO RESTORE RA BEFORE
010176  060
010177  113
                                                ; RETURNING TO THE MAIN PROGRAM
```

Figure D.4 (Continued)

```
                ;           LONG DELAY SUBROUTINE FOR R3 AS THE CALLING
                ;                       PROGRAM COUNTTER
                ;                 GENERATES 8N+22 CYCLES DELAY
                ;              R7 FUNCTIONS AS THE PROGRAM COUNTER

010341          EXDLY:  SEP     R3              ; RETURN TO THE CALLING PROGRAM
010341  323
010342          LDELAY: LDA     R3              ;
010342  103
010343                  PHI     R8              ; PUT HIGH AND LOW BYTES OF THE
010343  270
010344                  LDA     R3              ; DELAY CONSTANT INTO R8
010344  103
010345                  PLO     R8              ;
010345  250
010346          LDLY1:  DEC     R8              ; DECREMENT R8 IN A LOOP
010346  050
010347                  GHI     R8              ; HIGH BYTE OF R8-->D
010347  230
010350                  XRI     377             ; COMPLEMENT D
010350  373
010351  377
010352                  BNZ     LDLY1           ; BRANCH TO REPEAT IF NOT ZERO
010352  072
010353  346
010354                  BR      EXDLY           ; ELSE BRANCH TO ENTRY POINT MINUS 1
010354  060
010355  341
010356          END
```

Figure D.5 Long delay subroutine for R3 as the calling program counter.

```
                ;                TSE COMPUTER INPUT CONTROL PROGRAM
                ;              INSTRUCTION TIN.  REGISTER RD IS USED AS
                ;                        THE PROGRAM COUNTER

010262          EXTIN:  SEP     R3              ;
010262  323
010263          TINOP:  SEQ                     ; SET Q TO ENABLE INPUT IMAGE
010263  173
010264                  SEX     RD              ; PREPARE TO OUTPUT IMMEDIATE DATA
010264  355
010265                  OUT     3               ; TURN REGISTER A FEEDBACK PATH
010265  143
010266                  DB      <^B11100000>    ; REFORMATTER OFF
010266  340
010267                  MLDLY   747.            ; DELAY THREE GATE DELAYS
010267  171
010270  336
010271  002
010272  353
010273  042
010274                  OUT     3               ; TURN REGISTER A FEEDBACK PATH
010274  143
010275                  DB      <^B11110000>    ; REFORMATTER ON
010275  360
010276                  MLDLY   497.            ; DELAY TWO GATE DELAYS
010276  171
010277  336
010300  001
010301  361
010302  042
```

Figure D.6 Tse computer input control program.

```
010303                  REQ                     ; RESET Q TO DISABLE IMAGE INPUT
010303    172
010304                  B2      TIN1            ; CHECK FOR EXTERNAL INPUT REQUEST
010304    065
010305    313
010306                  OUT     5               ; IF THE REQUEST IS PRESENT ACKNOWLEDGE
010306    145
010307                  DB      <^B00000100>    ; BY SETTING BIT 3 OF PORT 5.  THE INPUT
010307    004
010310                  LBR     INPUT           ; ROUTINE MUST RESET THIS BIT.
010310    300
010311    017
010312    377
010313        TIN1:     B3      TIN2            ; CHECK FOR A REQUEST TO HALT TSE
010313    066
010314    317
010315                  BR      TIN1            ; PROCESSING AND GO INTO A PROGRAM
010315    060
010316    313
                                                ; LOOP IF THE REQUEST IS PRESENT
010317        TIN2:     SEX     R3              ; OTHERWISE RETURN THE VALUE OF X TO 3
010317    343
010320                  BR      EXTIN           ; AND BRANCH TO JUST BEFORE THE ENTRY
010320    060
010321    262
                                                ; POINT TO RESTORE RD BEFORE RETURNING
                                                ; TO THE MAIN PROGRAM.
```

Figure D.6 (Continued)

APPENDIX E

RT-11 MACRO ASSEMBLER

This appendix summarizes the features of the RT-11 Macro assembler [20] that are utilized in the tse computer cross-assembler. A detailed explanation of the Macro assembler can be found in [20].

The RT-11 Macro assembler offers three features that are essential to the tse computer cross-assembler. First, Macro permits user defined macros which allow new instructions to be defined. Second, Macro includes numerous conditional assembly directives which simplify the generation of special operational codes, and, third, Macro provides listing control directives which can be used to enhance the readability of assembled tse computer programs.

Macro accepts a source program with up to four fields. The general format of a source statement is

```
label:  operator operand(s) ; comments .
```

The label and comment fields are optional. Either the operator or the operand field may be omitted depending upon the contents of the other. When more than one operand appears in the operand field, the operands are separated by one of the legal separating characters defined in Table E.1. The legal character set for source statements includes the letters A through Z, the digits 0 through 9, and the special characters defined in Table E.2.

Some of the special characters listed in Table E.2 are used as operator characters which specify unary or binary operations on the given operands. Tables E.3 and E.4 define the legal unary and binary

TABLE E.1

LEGAL SEPARATING CHARACTERS [20]

Character	Definition	Usage
space	one or more spaces and/or tabs	A space is a legal separator only for argument operands. Spaces within expressions are ignored.
,	comma	A comma is a legal separator for both expressions and argument operands.
<...>	paired angle brackets	Paired angle brackets are used to enclose an argument, particularly when that argument contains separating characters. Paired angle brackets may be used anywhere in a program to enclose an expression for treatment as a term. (The angle bracket construction should be used when the argument contains unary operators).
↑\...\	Up arrow construction where the up arrow character is followed by an argument bracketed by any paired printing characters.	This construction is equivalent in function to the paired angle brackets and is generally used only where the argument contains angle brackets.

TABLE E.2

SPECIAL CHARACTERS [20]

Character	Designation	Function
carriage return line feed form feed vertical tab		formatting character source statement terminators
:	colon	label terminator
=	equal sign	direct assignment indicator
%	percent sign	register term indicator
tab		item or field terminator
space		item or field terminator
#	number sign	immediate expression indicator
@	at sign	deferred addressing indicator
(	left parenthesis	initial register indicator
)	right parenthesis	terminal register indicator
,	comma	operand field separator
;	semicolon	comment field indicator
<	left angle bracket	initial argument or expression indicator
>	right angle bracket	terminal argument or expression indicator
+	plus sign	arithmetic addition operator or auto increment indicator
-	minus sign	arithmetic subtraction operator or auto decrement indicator
*	asterisk	arithmetic multiplication operator
/	slash	arithmetic division operator
&	ampersand	logical AND operator
!	exclamation point	logical inclusive OR operator
"	double quote	double ASCII character indicator
'	single quote	single ASCII character indicator
↑	up arrow	universal unary operator, argument indicator
\	backslash	macro numeric argument indicator

TABLE E.3

OPERATOR CHARACTERS [20]

Unary Operator	Explanation		Example
+	plus sign	+A	(positive value of A, equivalent to A)
-	minus	-A	(negative, 2's complement value of A)
↑	up arrow, universal unary operator	↑F3.0	(interprets 3.0 as a 1-word floating-point number)
		↑C24	(interprets the one's complement of the binary representation of 24(8))
		↑D127	(interprets 127 as a decimal number)
		↑034	(interprets 34 as an octal number)
		↑B11000111	(interprets 11000111 as a binary value)

TABLE E.4

LEGAL BINARY OPERATORS [20]

Binary Operator	Explanation	Example	
+	addition	A+B	
-	subtraction	A-B	
*	multiplication	A*B	(16-bit product returned)
/	division	A/B	(16-bit quotient returned)
&	logical AND	A&B	
!	logical inclusive OR	A!B	

operators, respectively. Note the ↑O and ↑B constructions which are used extensively in the tse computer macros to indicate whether a number is in the octal or binary radix. Operands can be numbers or previously defined symbols. The symbols are usually defined by direct assignment statements which have the general format

symbol = expression.

The tse mask symbols, tse register symbols, and COSMAC register symbols are all defined by direct assignment statements. Their decimal values are given in Table 7.3, page 157.

In some instructions and macro definitions, the current value of the assembly location counter must be known. A special symbol, the period, is used to represent the current value of the assembly location counter. The period can be used in any expression in which the other defined symbols are legal. The tse computer cross-assembler uses the current value of the assembly location counter to check for illegal attempts to branch across page boundaries using short branch instructions. When an illegal branch is detected, a .ERROR directive is used to output an error message to the list file as a warning to the programmer. Error messages are also printed out when the programmer attempts to use an illegal input/output port or register specification.

The RT-11 Macro assembler provides several types of assembler directives which occupy the operator field of a Macro source line and cause the assembler to perform special processing operations. Listing control, data storage, terminating, conditional assembly, and macro directives are all used in the tse computer cross-assembler.

The listing control directives .LIST and .NLIST are used to control the contents of the list file created by the assembly process. Macro utilizes a listing level count to determine whether or not a particular line of source code should be listed. When used without an argument, the .LIST and .NLIST statements cause the listing level count to be incremented or decremented, respectively. Listing is suppressed whenever the listing level count is negative.

The listing control directives can also be used with arguments. In that case, the listing level count is not affected, but the listing mode is overridden in a manner specified by the argument. The most commonly used listing directive arguments are shown in Table E.5. Listing directives are used extensively in the tse computer cross-assembler to prevent macro expansions from listing. This improves the readability of the assembled applications programs.

Since the PDP 11/40 is a 16-bit minicomputer, the RT-11 Macro assembler normally works with 16-bit numbers. The COSMAC microprocessor, however, is an eight-bit machine that requires eight-bit source code and data bytes. A data storage directive, .BYTE, allows the Macro assembler to produce object files which are suitable for the COSMAC control unit of the tse computer. The .BYTE directive truncates specified arguments to eight bits. The argument can be a number or any legal expression whose 16-bit value has a high byte that contains either all zeros or all ones.

One terminating directive is used in the tse computer cross-assembler to indicate the physical end of a source program. This .END directive can have an optional argument that indicates the entry point of the

TABLE E.5

SOME ALLOWABLE LISTING DIRECTIVE ARGUMENTS [20]

Argument	Default	Function
SEQ	list	Controls the listing of source line sequence numbers.
LOC	list	Controls the listing of the location counter (this field would not normally be suppressed).
BIN	list	Controls the listing of generated binary code (supersedes BEX).
SRC	list	Controls the listing of the source code.
COM	list	Controls the listing of comments. This is a subset of the SRC argument and can be used to reduce listing time and/or space where comments are unnecessary.
SYM	list	Controls the listing of the symbol table for the assembly.

program. Often, this feature is used to provide automatic start-up of programs after they are loaded into the computer.

Conditional assembly directives are one of the most important Macro assembler features. These directives provide the programmer with the capability to conditionally include or ignore blocks of source code during the assembly process. The general form of a conditional block of Macro code is

```
.IF  condition,argument(s) ; Start of Conditional Block
  .                        ; Statements in the Range of
  .                        ; the Conditional Block
.ENDC                      ; End of Conditional Block
```

where the condition which must be met for the block to be included in the assembly is one of those given in Table E.6.

There are three subconditional directives (Table E.7) which can be placed within conditional blocks to indicate that an alternate section of code should be assembled when the main condition is not met. Alternately, these subconditionals can be used to indicate the unconditional assembly of a section of code within the conditional block. The value of the condition, found upon entering the conditional block of code, is the implied argument of the subconditional statements.

One line conditional blocks can be written using an immediate conditional directive of the form

```
.IIF condition, argument, statement.
```

The allowable conditions and arguments are the same as those defined earlier. Note that a .ENDC statement is not required for immediate conditionals.

TABLE E.6

ALLOWABLE CONDITIONS [20]

Conditions			
Positive	Complement	Arguments	Assemble Block If
EQ	NE	expression	expression=0 (or ≠ 0)
GT	LE	expression	expression>0 (or ≤ 0)
LT	GE	expression	expression<0 (or ≥ 0)
DF	NDF	symbolic argument	symbol is defined (or undefined)
B	NB	macro-type argument	argument is blank (or nonblank)
IDN	DIF	two macro-type arguments separated by a comma	arguments identical (or different)
Z	NZ	expression	same as EQ/NE
G		expression	same as GT/LE
L		expression	same as LT/GE

TABLE E.7

SUBCONDITIONAL DIRECTIVES [20]

Subconditional	Function
.IFF	The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is false.
.IFT	The code following this statement up to the next subconditional or end of the conditional block is included in the program if the value of the condition tested upon entering the conditional block is true.
.IFTF	The code following this statement up to the next subconditional or the end of the conditional block is included in the program regardless of the value of the condition tested upon entering the conditional block.

Macro directives are the final type of assembler directives utilized by the tse computer cross-assembler. A .MACRO statement of the form

.MACRO name, dummy argument list

serves as the first statement of each macro definition. The name of the macro can be any legal symbol. Similarly, any required arguments are represented by legal symbols in the dummy argument list. These symbols can be used outside the body of the macro definition with no conflicts of definition. A comment field can be included after the dummy argument list.

The last statement in each macro defintiion must be a .ENDM directive. The .ENDM directive is of the form

.END name

where name is an optional argument which, if used, must be the name of the macro being terminated. Examples of correctly defined and terminated macros are given in Appendix F.

APPENIDX F

SELECTED MACRO DEFINITIONS FROM THE TSE COMPUTER CROSS-ASSEMBLER MACRO LIBRARY

Figure F.1 is a listing of some representative macro definitions from the tse cross-assembler macro library. The macro library is intended to be used with Digital Equipment Corporation's RT-11 MACRO assembler and a PDP 11/40 minicomputer.

```
.MACRO SEP REG
.NLIST SRC
$$$ER REG
.BYTE <^0320+REG>
.LIST SRC
.ENDM SEP

.MACRO REQ
.NLIST SRC
.BYTE ^0172
.LIST SRC
.ENDM REQ

.MACRO  MLDLY   ADR
.NLIST SRC
.BYTE ^0171                                          ; MARK
.BYTE   ^0336,<ADR&^0177400>/^0400,ADR&^0377 ; SEP RE,HIADR,LADR
.BYTE ^0042                                          ; DEC R2
.LIST SRC
.ENDM MLDLY

.MACRO DLY6
.NLIST SRC
.BYTE ^0304,^0304 ; NOP NOP
.LIST SRC
.ENDM DLY6

.MACRO LDI RP
.NLIST SRC
.BYTE ^0370,RP
.LIST SRC
.ENDM LDI
```

Figure F.1 Representative macro definitions from the tse computer cross-assembler macro library.

```
.MACRO LDN REG
.NLIST SRC
$$$ER REG
.BYTE      REG
.LIST SRC
.ENDM LDN

.MACRO STR REG
.NLIST SRC
$$$ER REG
.BYTE <^O120+REG>
.LIST SRC
.ENDM STR

.MACRO  DB      X1,X2,X3,X4,X5,X6
.NLIST  SRC
.BYTE   X1
.IF     NB      X2
.BYTE   X2
.IF     NB      X3
.BYTE   X3
.IF     NB      X4
.BYTE   X4
.IF     NB      X5
.BYTE   X5
.IF     NB      X6
.BYTE   X6
.ENDC
.ENDC
.ENDC
.ENDC
.ENDC
```

Figure F.1 (Continued)

```
.MACRO B1 RP
.NLIST SRC
$$$PAG RP
.BYTE ^0064,RP&^0377
.LIST SRC
.ENDM B1

.MACRO BZ RP
.NLIST SRC
$$$PAG RP
.BYTE ^0062,RP&^0377
.LIST SRC
.ENDM BZ

.MACRO LBR ADR
.NLIST SRC
ADR1=<^0177400&ADR>/^0400
ADR2=ADR&^0377
.BYTE ^0300,ADR1,ADR2
.LIST SRC
.ENDM LBR

.MACRO PLO REG
.NLIST SRC
$$$ER REG
.BYTE <^0240+REG>
.LIST SRC
.ENDM PLO

.MACRO DEC REG
.NLIST SRC
$$$ER REG
.BYTE <^0040+REG>
.LIST SRC
.ENDM DEC
```

Figure F.1 (Continued)

```
.MACRO OUT REGL
.NLIST SRC
$$$ERL REGL
.BYTE <^0140+REGL>
.LIST SRC
.ENDM OUT

.MACRO TCLRI                    ; CLEAR REGISTER I
.NLIST SRC
.BYTE ^0144                     ; OUT 4
.BYTE ^B00000000                ; DB 0
.BYTE ^0327                     ; LDLY 494.
.DW   0756                      ; DELAY 2 GATE DELAYS
.LIST SRC                       ; THIS MACRO TURNS THE FEEDBACK PATH
.ENDM TCLRI                     ; REFORMATTER OF LATCH I OFF

.MACRO TCMR REG1,REG2,MASK      ; COMPLEMENT REGISTER
.NLIST SRC
.BYTE ^0331                     ; SEP R9
.IF EQ,REG2-^0341               ;
.IFT                            ; ASSEMBLE IF REG2 IS A
.BYTE MASK                      ; MASK CONTROL BYTE
.BYTE ^B10110001                ; REGISTER OUTPUT CONTROL BYTE
.IFF                            ; ASSEMBLE IF REG2 IS NOT A
.BYTE <MASK*16.>                ; MASK CONTROL BYTE
R0=<REG2&^0017>                 ; PICK OFF BOTTOM 4 BITS OF REG 2
.BYTE <^0220!R0>                ; REGISTER OUTPUT CONTROL BYTE
.ENDC
.BYTE ^B00000001                ; PORT SIX CONTROL BYTE.  P61 ON
.BYTE REG1                      ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^0360!REG1>              ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TCMR
```

Figure F.1 (Continued)

```
.MACRO TMOV REG1,REG2                ; MOVE REGISTER TO REGISTER
.NLIST SRC
.BYTE ^0331                          ; SEP R9
.BYTE ^B00000000                     ; MASK CONTROL BYTE
.IF EQ,REG2-^0341                    ;
.IFT                                 ; ASSEMBLE IF REG2 IS A
.BYTE ^B01100001                     ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000000                     ; PORT SIX CONTROL BYTE.   P61 OFF
.IFF                                 ; ASSEMBLE IF REG2 IS NOT A
R0=<REG2&^0017>                      ;
.BYTE<^020!R0>                       ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                     ; PORT SIX CONTROL BYTE.   P61 ON
.ENDC
.BYTE REG1                           ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^0360!REG1>                   ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TMOV

.MACRO TMVI REG,MASK                 ; MOVE IMMEDIATE TO REGISTER
.NLIST SRC
.BYTE ^0331                          ; SEP R9
.BYTE MASK                           ; MASK CONTROL BYTE
.BYTE ^B00000000                     ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                     ; PORT SIX CONTROL BYTE.   P61 ON
.BYTE REG                            ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^0360!REG>                    ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TMVI
```

Figure F.1 (Continued)

```
.MACRO TANDN REG1,REG2,MASK      ; AND NOT REGISTER TO A
.NLIST SRC
.BYTE ^O331                      ; SEP R9
.IF EQ,REG2-^O341
.IFT                             ; ASSEMBLE IF REG2 IS A
.BYTE MASK                       ; MASK CONTROL BYTE
.BYTE ^B01100001                 ; REGISTER OUTPUT CONTROL BYTE
.IFF                             ; ASSEMBLE IF REG2 IS NOT A
.BYTE <MASK&^O007>               ; MASK CONTROL BYTE
R0=<REG2&^O017>                  ; PICK OFF LOWER FOUR BITS OF REG2
.BYTE <^O140!R0>                 ; REGISTER OUTPUT CONTROL BYTE
.ENDC
.BYTE   ^B00000000               ; PORT SIX CONTROL BYTE.  P61 OFF
.BYTE REG1                       ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^O360!REG1>               ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TANDN

.MACRO TOR       REG,MASK        ; OR REGISTER TO A
.NLIST SRC
.BYTE ^O331                      ; SEP R9
.IF EQ,REG-^O341
.IFT                             ; ASSEMBLE IF REG IS A
.BYTE MASK                       ; MASK CONTROL BYTE
.BYTE ^B00100001                 ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000000                 ; PORT SIX CONTROL BYTE.  P61 OFF
.IFF                             ; ASSEMBLE IF REG IS NOT A
.BYTE <^O007&MASK>               ; MASK CONTROL BYTE
.BYTE <REG&^O017>                ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                 ; PORT SIX CONTROL BYTE.  P61 ON
.ENDC
.BYTE <^O360!REG>                ; REGISTER INPUT CONTROL BYTE NUMBER 1
.BYTE <^O360!REG>                ; REGISTER INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TOR
```

Figure F.1 (Continued)

```
.MACRO TCNT REG, MASK                ; CONTRACT REGISTER
.NLIST SRC
.BYTE ^O333                          ; SEP RC
.IF EQ, REG-^O341                    ;
.IFT                                 ; ASSEMBLE IF REG IS A
.BYTE MASK                           ; MASK CONTROL BYTE
.BYTE ^B00100001                     ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000000                     ; PORT SIX CONTROL BYTE.   P61 OFF
.IFF                                 ; ASSEMBLE IF REG IS NOT A
.BYTE <MASK&^O007>                   ; ALU MASK CONTROL BYTE
.BYTE <REG&^O177>                    ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000000 .                   ; PORT SIX CONTROL BYTE.   P61 ON
.ENDC
.LIST SRC
.ENDM TCNT


.MACRO TOUT REG                      ; OUTPUT TSE
.NLIST SRC
.BYTE ^O331                          ; SEP R9
.BYTE ^B00000000                     ; MASK CONTROL BYTE
R0=<REG&^O017>
.BYTE <^O020!R0>                     ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                     ; PORT SIX CONTROL BYTE.   P61 ON
.BYTE ^B01111000                     ; REGISTER 0 INPUT CONTROL BYTE NUMBER 1
.BYTE ^B11111000                     ; REGISTER 0 INPUT CONTROL BYTE NUMBER 2
.LIST SRC
.ENDM TOUT
```

Figure F.1 (Continued)

```
.MACRO TIDA X                    ; IDENTIFY BASIS POINTS WITH A
.NLIST SRC                       ; SURROUND OF INDEX X
.BYTE ^0332                      ; SEP RA
.IF EQ,X                         ; ASSEMBLE IF INDEX IS ZERO
.BYTE ^0001                      ; 1
.BYTE ^0277                      ; INDEX ZERO CONTROL BYTE
.ENDC
.IF EQ, X-^0006                  ; ASSEMBLE IF INDEX IS SIX
.BYTE ^0001                      ; 1
.BYTE ^0200                      ; INDEX SIX CONTROL BYTE
.ENDC
.IF EQ, X-^0007                  ; ASSEMBLE IF INDEX IS SEVEN
.BYTE ^0002                      ; 2
.BYTE ^0252, ^0225               ; INDEX SEVEN CONTROL BYTES
.ENDC
.IF GT, X-^0013                  ; ASSEMBLE IF INDEX IS TWELVE OR THIRTEEN
.BYTE ^0003                      ; 3
.IF EQ, X-^0014                  ; ASSEMBLE IF INDEX IS TWELVE
.BYTE ^0266, ^0255, ^0233        ; INDEX TWELVE CONTROL BYTES
.ENDC
.IF EQ, X-^0013                  ; ASSEMBLE IF INDEX THIRTEEN
.BYTE ^0211, ^0222, ^0244        ; INDEX THIRTEEN CONTROL BYTES
.ENDC
.ENDC
.IF NE, X                        ; ASSEMBLE IF INDEX IS NOT ZERO
.IF NE, X-^0006                  ; ASSEMBLE IF INDEX IS NOT SIX
.IF NE, X-^0007                  ; ASSEMBLE IF INDEX IS NOT SEVEN
.IF LE, X-^0013                  ; ASSEMBLE IF INDEX IS NOT TWELVE OR THIRTEEN
.BYTE ^0006                      ; 6
.IF EQ, X-^0001                  ; ASSEMBLE IF INDEX IS ONE
.BYTE ^0276, ^0275, ^0273        ; INDEX ONE CONTROL BYTES
.BYTE ^0267, ^0257, ^0237        ;
.ENDC
```

Figure F.1 (Continued)

```
.IF EQ, X-^0002                    ; ASSEMBLE IF INDEX IS TWO
.BYTE ^0274, ^0271, ^0263          ; INDEX TWO CONTROL BYTES
.BYTE ^0247, ^0217, ^0236
.ENDC
.IF EQ, X-^0003                    ; ASSEMBLE IF INDEX IS THREE
.BYTE ^0270, ^0261, ^0243          ; INDEX THREE CONTROL BYTES
.BYTE ^0207, ^0216, ^0234
.ENDC
.IF EQ, X-^0004                    ; ASSEMBLE IF INDEX IS FOUR
.BYTE ^0260, ^0251, ^0203          ; INDEX FOUR CONTROL BYTES
.BYTE ^0206, ^0215, ^0230
.ENDC
.IF EQ, X-^0005                    ; ASSEMBLE IF INDEX IS FIVE
.BYTE ^0240, ^0201, ^0202          ; INDEX FIVE CONTROL BYTES
.BYTE ^0204, ^0210, ^0220
.ENDC
.IF EQ, X-^010                     ; ASSEMBLE IF INDEX IS EIGHT
.BYTE ^0254, ^0231 ^0262           ; INDEX EIGHT CONTROL BYTES
.BYTE ^0245, ^0213, ^0226
.ENDC
.IF EQ, X-^0011                    ; ASSEMBLE IF INDEX IS NINE
.BYTE ^0264, ^0251, ^0223          ; INDEX NINE CONTROL BYTES
.BYTE ^0236, ^0215, ^0232
.ENDC
.IF EQ, X-^0012                    ; ASSEMBLE IF INDEX IS TEN
.BYTE ^0242, ^0250, ^0221          ; INDEX TEN CONTROL BYTES
.BYTE ^0224, ^0205, ^0212
.ENDC
.IF EQ, X-^013                     ; ASSEMBLE IF INDEX IS ELEVEN
.BYTE ^0253, ^0227, ^0256          ; INDEX ELEVEN CONTROL BYTES
.BYTE ^0235, ^0272, ^0265
.ENDC
.ENDC
```

Figure F.1 (Continued)

```
.ENDC
.ENDC
.ENDC
.LIST SRC
.ENDM TIDA

.MACRO TCMP REG, MASK              ; COMPARE REGISTER
.NLIST SRC
.BYTE ^0333                        ; SEP RC
M0=<MASK*16.>                      ; SHIFT LOWER FOUR BITS OF MASK BYTE INTO
M1=<M0!MASK>                       ; UPPER FOUR BITS AND COPY BACK INTO LOWER FOUR BITS ALSO
.BYTE <M1&^0167>                   ; MASK CONTROL BYTE
R0=<REG&^017>                      ; PICK OFF BOTTOM 4 BITS OF REG
.BYTE <R0!^0201>                   ; REGISTER OUTPUT CONTROL BYTE
.BYTE ^B00000001                   ; PORT SIX CONTROL BYTE.  P61 ON
.LIST SRC
.ENDM TCMP

.MACRO TIN                         ; INPUT TSE
.NLIST SRC
.BYTE ^0335                        ; SEP RD
.LIST SRC
.ENDM TIN

.MACRO TBNZ RP                     ; TSE BRANCH ON RF NOT EQUAL TO ZERO
.NLIST SRC
$$$PAG RP
.BYTE ^0217                        ; GLO RF
.BYTE ^0072, RP&^0377              ; BNZ RP
.LIST SRC
.ENDM TBNZ
```

Figure F.1 (Continued)

```
.MACRO $$$PAG RPT
.NLIST SRC
ADR1=<^O177400&RPT>/^O400
ADR2=<^O177400&.>/^O400
.IIF NE, <ADR1-ADR2>,    .ERROR; ^***: ILLEGAL BRANCH OVER PAGE BOUNDARY
.ENDM $$$PAG

.MACRO $$$ER REGT
$$$2=^O2
.IIF GE, REGT, $$$2=$$$2-1
.IIF LE, <REGT-15.>, $$$2=$$$2-1
.IIF NE, $$$2,  .ERROR;  ^***:  ILLEGAL REGISTER SPECIFICATION
.ENDM $$$ER

.MACRO $$$ERL REGLT
$$$2=2
.IIF GE, REGLT, $$$2=$$$2-1
.IIF LE, <REGLT-7.>, $$$2=$$$2-1
.IIF NE, $$$2,  .ERROR; ^***: ILLEGAL PORT ASSIGNMENT
.ENDM $$$ERL
```

Figure F.1 (Continued)

APPENDIX G

SAMPLE APPLICATIONS PROGRAMS FOR THE GOLAY TRANSFORM TSE COMPUTER

Figure G.1 is a listing of a program for performing the Golay transform skeletonizing algorithm using the tse computer. The skeletonizing program is 129 bytes long and requires 607 unit gate delays to process a simple image. A program for performing the swelling algorithm is listed as Figure G.2. Each iteration of the swelling algorithm takes 664 unit gate delays. The swelling program is 147 bytes long.

```
000100          START:   TIN                    ; INPUT NEXT IMAGE FOR SKELETONIZING
000100    335
000101          ANOTHER: TMOV     C,A           ; SAVE IMAGE IN REGISTER C
000101    331
000102    000
000103    141
000104    000
000105    264
000106    364
000107                   TCLRI                  ; CLEAR INDEX REGISTER
000107    144
000110    000
000111    327
000112    001
000113    356
000114                   TIDA     1             ; RECOGNIZE INDEX ONE
000114    332
000115    006
000116    276
000117    275
000120    273
000121    267
000122    257
000123    237
000124                   TIDA     2             ; RECOGNIZE INDEX TWO
000124    332
000125    006
000126    274
000127    271
000130    263
000131    247
000132    217
000133    236
```

Figure G.1 A program for performing the Golay transform skeletonizing algorithm.

```
000134                  TIDA    3               ; RECOGNIZE INDEX THREE
000134  332
000135  006
000136  270
000137  261
000140  243
000141  207
000142  214
000143  234
000144                  TANDN   A,I,M1          ; PERFORM GOLAY FUNCTION OPERATION
000144  331
000145  001
000146  150
000147  000
000150  341
000151  361
                                                ; FOR THE FIRST SUBFIELD
000152                  TCLRI                   ; CLEAR INDEX REGISTER
000152  144
000153  000
000154  327
000155  001
000156  356
000157                  TIDA    1               ; RECOGNIZE INDEX ONE
000157  332
000160  006
000161  276
000162  275
000163  273
000164  267
000165  257
000166  237
```

Figure G.1 (Continued)

```
000167                  TIDA    2               ; RECOGNIZE INDEX TWO
000167    332
000170    006
000171    274
000172    271
000173    263
000174    247
000175    217
000176    236
000177                  TIDA    3               ; RECOGNIZE INDEX THREE
000177    332
000200    006
000201    270
000202    261
000203    243
000204    207
000205    216
000206    234
000207                  TANDN   A,I,M2          ; PERFORM GOLAY FUNCTION OPERATION
000207    331
000210    002
000211    150
000212    000
000213    341
000214    361
                                                ; FOR THE SECOND SUBFIELD
000215                  TCLRI                   ; CLEAR INDEX REGISTER
000215    144
000216    000
000217    327
000220    001
000221    356
```

Figure G.1 (Continued)

```
000222                  TIDA    1               ; RECOGNIZE INDEX ONE
000222  332
000223  006
000224  276
000225  275
000226  273
000227  267
000230  257
000231  237
000232                  TIDA    2               ; RECOGNIZE INDEX TWO
000232  332
000233  006
000234  274
000235  271
000236  263
000237  247
000240  217
000241  236
000242                  TIDA    3               ; RECOGNIZE INDEX THREE
000242  332
000243  006
000244  270
000245  261
000246  243
000247  207
000250  214
000251  234
```

Figure G.1 (Continued)

```
000252                  TANDN   A,I,M3          ; PERFORM GOLAY FUNCTION OPERATION
000252    331
000253    007
000254    150
000255    000
000256    341
000257    361
                                                ; FOR THE THIRD SUBFIELD
000260                  TCMP    C,M             ; COMPARE ITERATION RESULT WITH IMAGE
000260    333
000261    104
000262    205
000263    001
                                                ; SAVED IN REGISTER C
000264                  TBNZ    ANOTHER         ; BRANCH TO PERFORM ANOTHER
000264    217
000265    072
000266    101
                                                ; ITERATION IF DIFFERENT
000267                  TOUT    A               ; OTHERWISE OUTPUT THE SKELETON IMAGE
000267    331
000270    000
000271    021
000272    001
000273    170
000274    370
000275          CHECK:  B2      START           ; CHECK FOR A NEW INPUT IMAGE
000275    045
000276    100
000277                  BR      CHECK           ; WAIT IN A LOOP FOR THE NEW IMAGE.
000277    060
000300    275
000301          END
```

Figure G.1 (Continued)

```
000400          BEGIN:  TIN                     ; INPUT IMAGE FOR SWELLING
000400  335
000401          CONTINUE:
000401                  TMOV    C,A             ; SAVE IMAGE IN REGISTER C
000401  331
000402  000
000403  141
000404  000
000405  264
000406  364
000407                  TCLRI                   ; CLEAR INDEX REGISTER
000407  144
000410  000
000411  327
000412  001
000413  356
000414                  TIDA    3               ; RECOGNIZE INDEX THREE
000414  332
000415  006
000416  270
000417  261
000420  243
000421  207
000422  216
000423  234
000424                  TIDA    4               ; RECOGNIZE INDEX FOUR
000424  332
000425  006
000426  260
000427  251
000430  203
000431  206
000432  215
000433  230
```

Figure G.2 A program for performing the Golay transform swelling algorithm.

```
000434                          TIDA    5               ; RECOGNIZE INDEX FIVE
000434  332
000435  006
000436  240
000437  201
000440  202
000441  204
000442  210
000443  220
000444                          TANDN   A,I,M1          ; PERFORM THE GOLAY FUNCTION OPERATION
000444  331
000445  001
000446  150
000447  000
000450  341
000451  361
000452                          TOR     I,M1            ; ON THE FIRST SUBFIELD IN TWO
000452  331
000453  001
000454  010
000455  001
000456  370
000457  370

                                                        ; OPERATIONS
000460                          TCLRI                   ; CLEAR INDEX REGISTER
000460  144
000461  000
000462  327
000463  001
000464  356
```

Figure G.2 (Continued)

```
000465                  TIDA    3               ; RECOGNIZE INDEX THREE
000465  332
000466  006
000467  270
000470  261
000471  243
000472  207
000473  216
000474  234
000475                  TIDA    4               ; RECOGNIZE INDEX FOUR
000475  332
000476  006
000477  260
000500  251
000501  203
000502  206
000503  215
000504  230
000505                  TIDA    5               ; RECOGNIZE INDEX FIVE
000505  332
000506  006
000507  240
000510  201
000511  202
000512  204
000513  210
000514  220
```

Figure G.2 (Continued)

```
000515                  TANDN   A,I,M2          ; PERFORM THE GOLAY FUNCTION OPERATION
000515  331
000516  002
000517  150
000520  000
000521  341
000522  361
000523                  TOR     I,M2            ; ON THE SECOND SUBFIELD IN TWO
000523  331
000524  002
000525  010
000526  001
000527  370
000530  370
                                                ; OPERATIONS
000531                  TCLRI                   ; CLEAR INDEX REGISTER
000531  144
000532  000
000533  327
000534  001
000535  356
000536                  TIDA    3               ; RECOGNIZE INDEX THREE
000536  332
000537  006
000540  270
000541  261
000542  243
000543  207
000544  216
000545  234
```

Figure G.2 (Continued)

```
000546                  TIDA    4               ; RECOGNIZE INDEX FOUR
000546  332
000547  006
000550  260
000551  251
000552  203
000553  206
000554  215
000555  230
000556                  TIDA    5               ; RECOGNIZE INDEX FIVE
000556  332
000557  006
000560  240
000561  201
000562  202
000563  204
000564  210
000565  220
000566                  TANDN   A,I,M3          ; PERFORM GOLAY FUNCTION OPERATION
000566  331
000567  007
000570  150
000571  000
000572  341
000573  361
000574                  TOR     I,M3            ; ON THE THIRD SUBFIELD IN TWO
000574  331
000575  007
000576  010
000577  001
000600  370
000601  370
                                                ; OPERATIONS
```

Figure G.2 (Continued)

```
000602                          TCMP    C,M             ; COMPARE RESULT TO IMAGE STORED IN
000602  333
000603  104
000604  205
000605  001
                                                        ; REGISTER C
000606                          TBNZ    CONTINUE        ; BRANCH TO PERFORM ANOTHER ITERATION
000606  217
000607  072
000610  001
                                                        ; IF DIFFERENT
000611                          TOUT    A               ; IF NO DIFFERENCE OUTPUT THE SWELLED
000611  331
000612  000
000613  021
000614  001
000615  170
000616  370
                                                        ; IMAGE
000617          CHECK:  B2      BEGIN                   ; CHECK FOR NEW IMAGE INPUT REQUEST AND
000617  065
000620  000
000621                          BR      CHECK           ; CONTINUE CHECKING IN A LOOP UNTIL ONE
000621  060
000622  217
                                                        ; OCCURS
000623                          END
```

Figure G.2 (Continued)